IBM® Rational® Functional TesterHCL OneTest™ UI

10.5.3 Documentation

June 2023

Special notice

Before using this information and the product it supports, read the information in Notices on page mcmxviii.

Contents

Chapter 1. Release Notes....9
Product description....9
What's New....9
Deprecated capabilities....10
Removed capabilities....11
Installing the product....11
Known issues....11
Contacting IBM Rational Software SupportContacting HCL support....12
Chapter 2. System Requirements....13
Hardware....15
Operating systems....16
Host prerequisites....19
Recording and playback support....22
Supported software....28
Chapter 3. Getting Started Guide....32
Rational® Functional TesterHCL OneTest™ UI overview....32
Task flow for Setting up Rational® Functional TesterHCL OneTest™ UI....36
Creating a project....38
Getting started in the UI Test perspective....38
Task flow for testing web applications....38
Task flows for testing mobile applications....41
Task flows for testing Windows desktop applications....42
Task flow for standard accelerated functional testing....43
Task flow for advanced accelerated functional testing....44
Getting started in the Functional Test perspective....45
The Functional Test perspective....46
Importing a sample functional test project....47
Task flow for testing applications....47
Task flow for testing Java applications....49
Task flow for testing HTML applications....51
Task flow for testing Eclipse applications....53
Task flow for testing SAP applications....55
Task flow for testing Flex applications....57
Visual Studio .NET window....59
Using the Functional Test perspective of on Linux....60
Task flow for integrating Rational® Functional TesterHCL OneTest™ UI and Rational® Integration TesterHCL OneTest™ API....63
Chapter 4. Tutorials....65
Tutorials for testing in the UI Test perspective....65
Lesson 1: Recording actions on an AUT....66
Lesson 2: Creating a verification point....68
Lesson 3: Running a test....70
Lesson 4: Test log and reports....72
Lesson 5: Splitting actions in a test....76
Lesson 6: Creating a dataset....81
Lesson 7: Associating a dataset with a test....82

Lesson 8: Creating a compound test ... 86
Lesson 9: Adding a loop ... 88
Tutorials for testing in the Functional Test perspective ... 90
Get started with functional testing using simplified scripts ... 90
Create a functional test using Java scripts ... 104
Perform a data-driven functional test using Java scripts ... 124
Test Adobe Flex application ... 134
Test GEF applications ... 138
Extend Rational® Functional TesterHCL OneTest™ UI capabilities using Proxy SDK ... 142
Tutorials for testing in the Visual Studio IDE ... 147
Create a functional test ... 147
Create a data-driven functional test ... 166
Test Adobe Flex application ... 175
Chapter 5. Samples ... 180
Sample project to test a Java application ... 180
Functional testing of a Java application sample project ... 181
A Rational® Functional TesterHCL OneTest™ UI project to test an HTML application ... 183
Testing the sample ... 184
Functional testing proxy SDK technology samples ... 184
ButtonProxy ... 184
JFormattedTextFieldProxy ... 186
CheckBoxProxy ... 188
Button OverrideProxy ... 191
JSpinnerProxy ... 193
TreeProxy ... 194
Flex control proxy ... 196
Functional testing samples for the Visual Studio IDE ... 198
Functional testing proxy SDK samples for the Visual Studio IDE ... 199
Chapter 6. Administrator Guide ... 201
Installation ... 201
Installation requirements ... 201
Installing the product using Installation Manager ... 202
Installation of the product by using the stand-alone installer ... 230
Uninstallation of the product by using the stand-alone installer ... 284
Upgrading and migrating ... 293
License management ... 296
License description ... 297
License configuration ... 297
Collecting usage metrics data ... 301
License management ... 302
License descriptions ... 302
Runtime licenses enablement for testing ... 304
Viewing license information for installed packages ... 304
Starting Rational® Functional TesterHCL OneTest™ UI from the command line ... 304
Software configuration management ... 305
Integrations in UI Test perspective ... 306
Integration plugin compatibility matrix ... 306

Testing with Ant....308
Testing with Azure DevOps for UI tests....317
Testing with EGit....336
Configuration of test runs as a GitHub Actions Workflow....346
Testing with IBM® Engineering Workflow Management....355
Testing with Engineering Test Management....357
Testing with Jenkins....368
Testing with Maven....393
Testing with Micro Focus ALM....404
Testing with Rational® Integration TesterHCL OneTest™ API....418
Testing with HCL® Launch....427
Testing with UrbanCode Deploy....448
Integrations in Functional Test perspective....459
Integration plugin compatibility matrix....459
Testing with Ant....461
Testing with Azure DevOps for functional tests....462
Testing with Cucumber....471
Testing with Micro Focus Application Life Cycle Management....480
Configuration of test runs as a GitHub Actions Workflow....481
Testing with IBM® Engineering Test Management....484
Testing with IBM® Engineering Test Management in the Visual Studio IDE....501
Testing with Jenkins....501
Testing with Maven....511
Testing with IBM® Engineering Workflow Management....514
Testing with Tivoli Composite Application Manager....517
Testing with HCL® Launch....518
Testing with UrbanCode™ Deploy....529
Chapter 7. Test Author Guide....532
Testing in the UI Test perspective....532
Testing web applications....532
Testing mobile applications....687
Testing Windows desktop applications....707
Recording SAP tests....714
Working with Selenium or Appium tests....733
Compound tests....737
Accelerated Functional Tests....745
Working with keywords....770
Testing in the Functional Test perspective....773
Preparing the functional test environment....774
Managing functional test projects....858
Functional test assets....864
Working with functional test scripts (Windows-only)....865
Verification points....904
Driving functional tests with external data....937
Testing terminal-based applications....956
Troubleshooting issues....1042
Extending the UI Test perspective with custom code....1042

Creating custom Java™ code....1042
Test execution services interfaces and classes....1044
Reducing the performance impact of custom code....1047
Custom code examples....1048
Adding custom JavaScript code as a test step in a Web UI test....1075
Extending the Functional Test perspective....1077
Rational® Functional TesterHCL OneTest™ UI proxy SDK....1078
Customizing a script template....1166
Using the API to edit functional test scripts....1179
Extending the functionality in the Visual Studio IDE....1234
Experimental Features....1234
Chapter 8. Test Execution Specialist Guide....1235
Configuration of test runs from the UI Test perspective....1235
Configuration of Web UI test runs....1235
Configuration of mobile test runs....1277
Configuration of Windows test runs....1350
Configuration of AFT Suite runs....1352
Running a test on multiple browsers and devices simultaneously....1376
Setting a global delay for test steps....1380
Configuration of test runs from the command line....1381
Configuration of test runs from the Functional Test perspective....1434
Script Playback....1436
Restoring the test environment before playback....1437
Configuring how to handle unexpected windows during playback....1438
Inserting dynamic test objects....1439
Enabling the dynamic find feature....1440
Enabling the dynamic find feature in the Visual Studio IDE....1441
Using ScriptAssure....1441
Ambiguous object recognition in functional testing....1443
Playback Monitor....1443
Pausing or stopping script playback....1444
Running scripts from Rational® Functional TesterHCL OneTest™ UI....1445
Playback of test scripts for HTML applications by using the Web UI engine....1447
Running a script from the Microsoft™ Edge browser....1449
Debugging scripts....1449
Screen snapshot on playback failure of functional tests....1450
Chapter 9. Test Manager Guide....1452
Publishing test result to IBM® Rational® Test Automation ServerHCL OneTest™ Server....1452
Publishing specific results to the server....1453
Unified reports....1455
Unified reports for Web UI, mobile, and Windows tests....1456
Unified reports for functional test scripts....1459
Prerequisites to export unified reports as a PDF....1463
Customizing PDF files exported from unified reports....1464
Removing customization of PDF files exported from unified reports....1467
Exporting unified reports....1468
Results for tests in UI Test perspective....1469

UI Test Statistical report....1469
Evaluating desktop Web UI results....1471
Customizing reports....1479
Export test results....1487
Evaluating mobile test run results....1491
Logs overview....1491
Results for tests in Functional Test perspective....1497
Functional test logs....1497
Logging page....1498
Setting log preferences....1500
Disabling enhanced log results....1501
Viewing logs in the Projects view....1501
Viewing Dojo logs....1502
Renaming and deleting logs....1502
Log Extension....1502
Chapter 10. Troubleshooting....1507
Troubleshooting in the UI Test perspective....1507
Unable to play back Web UI tests when certain web applications are redirected to a different URL....1507
Unable to open a browser for the Web UI test playback....1507
The AUT displays the 403 forbidden error....1508
Unable to verify content in a Windows test....1508
Unable to run Web UI tests on a remote agent on Linux....1509
Unable to run Web UI tests on an Android 13 device or emulator....1509
Unable to generate tests in a Linux or Mac operating system....1510
Rational® Functional TesterHCL OneTest™ UI error messages....1510
Frequently Asked Questions....1538
Troubleshooting in the Functional Test perspective....1538
Troubleshooting functional tests in Mozilla Firefox browsers....1539
Unable to test eclipse-based applications....1541
Ambiguous object recognition in functional testing....1542
Screen snapshot on playback failure of functional tests....1542
Tips and tricks for functional testing HTML applications....1543
Java applets in HTML pages....1547
Standard properties available for functional testing HTML objects....1548
Uninstalling Rational® Functional TesterHCL OneTest™ UI cleanly....1552
Problems with object recognition....1553
Unable to find the correct control by using the Insert Test Object option....1555
Unable to create a verification point by using the vpDynamic action....1556
Troubleshooting issues in SAP tests....1556
Problems with environment enablement....1557
Handling exceptions....1558
Collecting Rational® Functional TesterHCL OneTest™ UI error logs....1559
Viewing trace files within Rational® Functional TesterHCL OneTest™ UI....1560
AutoPD Collection artifacts in IBM® Support Assistant....1560
Frequently asked questions....1561
Rational® Functional TesterHCL OneTest™ UI error messages....1572
Chapter 11. Reference Guide....1699

Reference information for the UI Test perspective....1699
Mobile test preferences....1699
Mobile test reference....1700
Reference information for the Functional Test perspective....1703
Test application domain support....1703
Command line interface....1726
UI reference....1733
Reference information for the Visual Studio IDE....1893
VB.NET scripting command-line interface....1893
Test application domain support....1896
Add New Item dialog box....1898
dataset Literal Substitution dialog box....1899
Development Environment options....1900
Menus....1902
Functional Test project properties page....1908
Functional Test script properties page....1909
Toolbar....1910
Options dialog box....1914
Java API reference....1916
Security Considerations....mcmxvii
Security Considerations for IBM® Rational® Functional TesterHCL OneTest™ UI....mcmxvii
Notices....mcmxviii
Index....1923

Chapter 1. Release notes for IBM® Rational® Functional TesterHCL OneTest™ UI

This document contains information about what's new, installation instructions, known problems in and contact information of IBMHCL Customer Support.

Contents

- Product description on page 9
- What's New on page 9
- Deprecated capabilities on page 10
- Removed capabilities on page 11
- Installing the product on page 11
- Known issues on page 11
- Contacting IBM Rational Software Support on page 12Contacting HCL Support on page 12

Product description

IBM® Rational® Functional TesterHCL OneTest™ UI is a GUI test automation tool that is used to accelerate the functional and regression testing. It provides a flexible environment to test the desktop and web applications very efficiently. It is used to test a wide range of applications that includes HTML, web, native and hybrid mobile, SAP, Java, Power Builder, and Windows desktop applications. Rational® Functional TesterHCL OneTest™ UI is available in two integrated development environments (IDE): Eclipse and Microsoft Visual Studio .NET. The Eclipse integration supports both Java and simplified (non-programming) method of scripting while the Microsoft Visual Studio .NET integration supports Microsoft Visual Basic .NET scripting language. See Product overview on page 32.

What's New

- **Customizing names of exported statistical reports**

  You can now specify a pattern that defines the file names of the exported statistical reports when you run tests from the command line. See Configuration of test runs from the command line on page 1388.

- **Support for BrowserStack Cloud**

  You can now run Web UI and Mobile tests on Android or iOS devices that are connected to the BrowserStack Cloud testing platform. Also, you can use the command line to run tests on the BrowserStack Cloud. See the following links:
  - Creating an AFT Suite to run Web UI tests on BrowserStack Cloud on page 747
  - Creating an AFT Suite to run mobile tests on BrowserStack Cloud on page 757

- Running Web UI tests on BrowserStack Cloud on page 1248
- Running mobile tests on Android devices on BrowserStack Cloud on page 1286
- Running mobile tests on iOS devices on BrowserStack Cloud on page 1317
- Running compound tests on Android devices on BrowserStack Cloud on page 1302
- Running compound tests on iOS devices on BrowserStack Cloud on page 1331
- Running Web UI tests as an AFT Suite on BrowserStack Cloud on page 1365
- Running mobile tests as an AFT Suite on BrowserStack Cloud on page 1342
- Running Web UI tests on BrowserStack Cloud from the command-line on page 1404
- Running mobile tests for Android applications on BrowserStack Cloud from the command-line on page 1419
- Running mobile tests for iOS applications on BrowserStack Cloud from the command-line on page 1428

- **Support for MacOS Ventura 13.x**

  You can now install Rational® Functional TesterHCL OneTest™ UI by using Installation Manager on MacOS Ventura 13.x operating systems and run tests.

- **Support for Revenera 2022.12**

  With Revenera 2018.02 (LLS 1.0) reaching EOS, you can now use Revenera 2022.12 (LLS 3.0) as a local license server to host licenses for Rational® Functional TesterHCL OneTest™ UI.

- **Support to control test runs for error handling**

  A new error condition Test Verdict Failed is added to Error handling to manage the test when the overall verdict of the test is failed. The error condition can control the behavior of the test and can contribute to the health of the test. See Error conditions.

- **Support to run functional tests with JavaScript dialogs recorded on Internet Explorer**

  You can now play back functional test steps with JavaScript dialogs that were recorded on Internet Explorer by using the Web UI engine on the supported browsers.

  Also, you can write the functional test scripts to test JavaScript dialogs by using the find API function. See Dynamic Find() API for JavaScript dialogs on page 1231.

Deprecated capabilities

You can find information about the capabilities that are deprecated in this release and are planned for a removal in a future release.

Note: You can contact the IBM SupportHCL Client Support team if you have any concerns.

- No capabilities were deprecated in this release.

Removed capabilities

No capabilities were removed in this release.

Installing the product

For installation instructions, see Installing Rational Functional TesterHCL OneTest UI on page 212.

Remember:

- You can upgrade to Rational® Functional TesterHCL OneTest™ UI 10.5.3 from the previous versions, 10.5.2, 10.5.1, and 10.5.0. See Updating Rational Functional TesterHCL OneTest UI on page 293.
- If you installed Rational® Functional TesterHCL OneTest™ UI that is earlier than 10.5.0, then you cannot upgrade the product to the latest version. You must uninstall the existing version of the product before you install the latest version of the product.
- After you install Rational® Functional TesterHCL OneTest™ UI 10.5.3, at any point in time if you want to use the previous version of the product, you cannot roll back to the previous version. If you want to use the previous version of the product, you must uninstall the existing version, and then install the required version of the product.

Remember: You cannot upgrade the product to the latest version. You must uninstall the existing version of the product before you install the latest version of the product.

Known issues

You can find information about the known issues identified in this release of Rational® Functional TesterHCL OneTest™ UI.

Product	Download document	Knowledge base
Rational® Functional TesterHCL OneTest™ UI	Release document Release document	Knowledge articlesKnowledge articles

Known problems are documented in the download document and in the form of individual tech notes in the Support Knowledge Base. See the Support Knowledge Base.

The knowledge base is continually updated as issues are discovered and resolved. By searching the knowledge base, you can quickly find workarounds or solutions to issues.

Contacting IBM Rational Software SupportContacting HCL support

- For contact information and guidelines or reference materials that you might need when you require support, read the IBM Support Guide.
- For personalized support that includes notifications of significant upgrades, subscribe to Product notification.
- For technical assistance, contact HCL Customer Support.
- Before you contact IBM Rational Software SupportHCL support, you must gather the background information that you might need to describe your problem. When you describe a problem to an IBM softwarethe HCL support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:
    - What software versions were you running when the problem occurred?
    - Do you have logs, traces, or messages that are related to the problem?
    - Can you reproduce the problem? If so, what steps do you take to reproduce it?
    - Is there a workaround for the problem? If so, be prepared to describe the workaround.

Chapter 2. System Requirements

This document includes information about hardware and software requirements for IBM® Rational® Functional TesterHCL OneTest™ UI.

Contents

- Hardware on page 15
    - Linux on page 15
    - Mac on page 15
    - Windows on page 16
- Operating systems on page 16
    - Linux on page 17
    - Mac on page 17
    - Windows on page 18
    - Mobile on page 19
- Host prerequisites on page 19
    - Development tools on page 20
    - Licensing on page 20
    - Web browsers on page 21
    - Virtualization management on page 22
- Recording and playback support on page 22
    - Eclipse runtime environment on page 23
    - Java SDK on page 23
    - Product Specific or Mixed Content on page 24
    - Terminal emulation on page 26
    - Web browsers on page 26
- Supported software on page 28
    - Development tools on page 28
    - Device clouds on page 29
    - Integration Middleware on page 30

Disclaimers

This report is subject to the Terms of Use and the following disclaimers:

The information contained in this report is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied, including but not limited to the implied warranties of merchantability, non-infringement, and fitness for a particular purpose. In addition, this information is based on HCL's current product plans and strategy, which are subject to change by HCL without notice. HCL shall not be responsible for any direct, indirect, incidental, consequential, special or other damages arising out of the use of, or otherwise related to, this report or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating

any warranties or representations from HCL or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of HCL software.

References in this report to HCL products, programs, or services do not imply that they will be available in all countries in which HCL operates. Product release dates and/or capabilities referenced in this presentation may change at any time at HCL's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Discrepancies found between reports and other HCL documentation sources may or may not be attributed to different publish and refresh cycles for this tool and other sources. Nothing contained in this report is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth, savings or other results. You assume sole responsibility for any results you obtain or decisions you make as a result of this report.

Notwithstanding the Terms of Use users of this site are permitted to copy and save the reports generated from this tool for such users own internal business purpose. No other use shall be permitted.

Disclaimers

This report is subject to the Terms of Use (https://www.ibm.com/legal/us/en/) and the following disclaimers:

The information contained in this report is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied, including but not limited to the implied warranties of merchantability, non-infringement, and fitness for a particular purpose. In addition, this information is based on IBM’s current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any direct, indirect, incidental, consequential, special or other damages arising out of the use of, or otherwise related to, this report or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this report to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. The underlying database used to support these reports is refreshed on a weekly basis. Discrepancies found between reports generated using this web tool and other IBM documentation sources may or may not be attributed to different publish and refresh cycles for this tool and other sources. Nothing contained in this report is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth, savings or other results. You assume sole responsibility for any results you obtain or decisions you make as a result of this report.

Notwithstanding the Terms of Use (https://www.ibm.com/legal/us/en/), users of this site are permitted to copy and save the reports generated from this tool for such users own internal business purpose. No other use shall be permitted.

Hardware

You can find information about the hardware requirements for IBM® Rational® Functional TesterHCL OneTest™ UI.

Contents

- Linux on page 15
- Mac on page 15
- Windows on page 16

Linux

Hardware	Requirement	Notes
Disk space	10 GB	• Long test runs require large amount of disk space. You must ensure that there is adequate disk space before running tests. • Disk space consumption can be reduced by selecting the degree of logging in a test run. • For best results with large projects, long test runs, or AFT test runs, you must use 16 GB RAM. • For best results with long test runs or large AFT runs, you must use 2.30 GHz or higher processor.
Memory	8 GB	
Processor	Any modern processor that supports the operating system.	

Mac

Hardware	Requirement	Notes
Disk space	10 GB	• Long test runs require large amount of disk space. You must ensure that there is adequate disk space before running tests. • Disk space consumption can be reduced by selecting the degree of logging in a test run. • For best results with long test runs, you must use 2.30 GHz or higher processor.
Memory	16 GB	
Processor	Any modern processor that supports the operating system.	

Windows

Hardware	Requirement	Notes
Disk space	10 GB	• Long test runs require large amount of disk space. You must ensure that there is adequate disk space before running tests. • Disk space consumption can be reduced by selecting the degree of logging in a test run. • For best results with large projects, long test runs, or AFT test runs, you must use 16 GB RAM. • For best results with long test runs or large AFT runs, you must use 2.30 GHz or higher processor.
Memory	8 GB	
Processor	Any modern processor that supports the operating system.	

Operating systems

You can find the operating systems that are supported, organized by operating system family for Rational® Functional TesterHCL OneTest™ UI.

Contents

- Linux on page 17
- Mac on page 17
- Windows on page 18
- Mobile on page 19

Bit version support

Different parts of a product might run on the same operating system but support different application bitness.

For example one part of the product might run only in 32-bit mode, whereas another might support 64-bit tolerate mode.

Bitness	Description
32	The product or part of the product runs as a 32-bit application in the 32-bit platforms listed as supported.
64-Tolerate	The product or part of the product runs as a 32-bit application in the 64-bit platforms listed as supported.

Bitness	Description
64-Exploit	The product or part of the product runs as a 64-bit application in the 64-bit platforms listed as supported.

Linux

Operating system	Hardware	Bitness	Perspective		Notes
			Function-al Test	UI Test	
Red Hat Enterprise Linux (RHEL) 7.9	x86-64	64-Exploit	✔	✔	
Red Hat Enterprise Linux (RHEL) 8.4	x86-64	64-Exploit	✔	✔	Deprecated in 10.5.2.
Red Hat Enterprise Linux (RHEL) 8.6	x86-64	64-Exploit	✔	✔	
Red Hat Enterprise Linux (RHEL) 9.0	x86-64	64-Exploit	✔	✔	
Ubuntu 20.04 LTS	x86-64	64-Exploit	✔	✔	
Ubuntu 22.04 LTS	x86-64	64-Exploit	✔	✔	

Mac

Operating system	Hardware	Bitness	Perspective		Notes
			Function-al Test	UI Test	
macOS BigSur 11.6	x86-64	64-Exploit	✘	✔	Requires a minimum of 16 GB memo-

Operating system	Hardware	Bitness	Perspective		Notes
			Function-al Test	UI Test	
					ry for best results. Deprecated in 10.5.3.
macOS Monterey 12	x86-64	64-Exploit	✗	✓	Supports installation only by using Installation Manager.
MacOS Ventura 13	x86-64	64-Exploit	✗	✓	

Windows

Operating system	Hardware	Bitness	Perspective		Notes
			Function-al Test	UI Test	
Windows 10 Enterprise	x86-64	32, 64-Exploit	✓	✓	
Windows 10 Pro	x86-64	32, 64-Exploit	✓	✓	
Windows 11 Pro	x86-64	32, 64-Exploit	✓	✓	
Windows 11 Enterprise	x86-64	32, 64-Exploit	✓	✓	
Windows Server 2016	x86-64	32, 64-Exploit	✓	✓	Deprecated in 10.5.0.

Operating system	Hardware	Bitness	Perspective		Notes
			Function-al Test	UI Test	
Windows Server 2019	x86-64	32, 64-Ex-ploit	✔	✔	
Windows Server 2022	x86-64	32, 64-Ex-ploit	✔	✔	

Mobile

Operating system	Mobile tests	Notes
Android 11	✔	Deprecated in 10.5.3.
Android 12	✔	
Android 12L	✔	
Android 13	✔	
iOS 14	✔	Deprecated in 10.5.3.
iOS 15	✔	

Host prerequisites

You can find the host prerequisites that support the operating capabilities for IBM® Rational® Functional TesterHCL OneTest™ UI.

Contents

- Development Tools on page 20
- Licensing on page 20
- Licensing on page 21

- Web browsers on page 21
- Virtualization management on page 22

Development tools

Development Tool	Version	Perspective		Notes
		Functional Test	UI Test	
Appium	1.22	✗	✓	
Microsoft .NET Framework	3.5	✓	✓	
Microsoft Visual Studio	2015 and future fix packs	✓	✗	Deprecated in 10.5.0. Required to use IBM® Rational® Functional TesterHCL OneTest™ UI from this Integrated Development Environment (IDE).
	2017 and future fix packs	✓	✗	Required to use IBM® Rational® Functional TesterHCL OneTest™ UI from this Integrated Development Environment (IDE).
	2019 and future fix packs	✓	✗	

Licensing

License Server	Version	Perspective	
		Functional Test	UI Test
IBM Common License Key Server	9	✓	✓

Licensing

License Server	Version	IBM® Rational® Functional TesterHCL OneTest™ UI
HCL Local License Server	2.0	✔
HCL Local License Server	3.0	✔

Web browsers

The browsers listed in the following table can be used to view the contents of various assets in a testing project such as a recording session, configured applications, datasets, reports, and so on. You must refer to the Recording and Playback Support on page 26 section to view the list of browsers along with their versions that are supported to record and play back tests.

Browser	Version	Perspective		Notes
		Functional Test	UI Test	
Apple Safari	15	Not Applicable	✔	
	16	Not Applicable	✔	
Google Chrome	104 to 106	✔	✔	Deprecated in 10.5.3.
	107 to 114	✔	✔	
Microsoft Edge	104 to 105	✔	✔	Deprecated in 10.5.3.
	106 to 113	✔	✔	
Mozilla Firefox (including ESR versions)	91 (Only ESR)	✔	✔	Deprecated in 10.5.0.
	102	✔	✔	
	103 to 105	✔	✔	Deprecated in 10.5.3.

Browser	Version	Perspective		Notes
		Functional Test	UI Test	
	106 to 113	✔	✔	
Opera	89 to 90	✔	✔	Deprecated in 10.5.3.
	92 to 97	✔	✔	

Virtualization management

Container	Version	Perspective		Notes
		Functional Test	UI Test	
Docker Engine	20.1, 23, and future fix packs	✔	✔	
Docker Compose	2.10	✔	✔	Deprecated in 10.5.3.
	2.12, 2.15.0, and 2.17.0 future fix packs	✔	✔	In Docker Compose, the HTML test scripts that are created in the Functional Test perspective are supported only as part of a compound test in the UI Test perspective.

Recording and playback support

The recording and playback support section specifies the software that IBM® Rational® Functional TesterHCL OneTest™ UI supports to record and play back tests.

Contents

- Eclipse runtime environment on page 23
- Java SDK on page 23
- Product Specific or Mixed Content on page 24
- Terminal emulation on page 26
- Web browsers on page 26
- Mobile applications on page 28

Eclipse runtime environment

Supported software	Version	Functional Test perspective		Notes
		Record	Playback	
Eclipse	4.7 to 4.8	✓	✓	Deprecated in 10.5.0.
	4.23	✓	✓	
	4.26	✓	✓	
	4.27	✓	✓	

Java SDK

Supported software	Version	Functional Test perspective		Notes
		Record	Playback	
IBM Java	8	✓	✓	To test applications based on IBM Java.
Oracle Java	8	✓	✓	To test applications based on Oracle Java.
	11	✓	✓	
	17	✓	✓	
OpenJDK	8	✓	✓	To test applications based on OpenJDK.

Supported software	Version	Functional Test perspective		Notes
		Record	Playback	
	11	✓	✓	
	17	✓	✓	

Product Specific or Mixed Content

Supported software	Version	Perspective				Notes
		Functional Test		UI Test		
		Record	Playback	Record	Playback	
Adobe Flex SDK	4.5	✓	✓	✗	✗	Deprecated in 10.2.0.
Adobe Reader	XI	✓	✓	✗	✗	
HTML Version	4	✓	✓	✓	✓	Also supported for the playback of mobile tests.
	5	✗	✗	✓	✓	
Microsoft Visual Basic	6	✓	✓	✓	✓	In the UI Test perspective, this is supported by the Windows Test type.
SAP GUI	7.6	✓	✓	✓	✓	Deprecated in 10.2.3.
	7.7	✗	✗	✓	✓	To record and play back tests of SAP applications built with the SAP GUI client.

Supported software	Version	Perspective				Notes
		Functional Test		UI Test		
		Record	Playback	Record	Playback	
						Supported via Rational® Performance TesterHCL OneTest™ Performance extension.
Power Builder	12.6	✓	✓	✓	✓	Support for functional tests is deprecated in 10.5.0. In the UI Test perspective, this is supported by the Windows Test type.
	2017	✓	✓	✗	✗	
.Net applications	4.6.1	✓	✓	✓	✓	Support for functional tests is deprecated in 10.5.0. In the UI Test perspective, this is supported by the Windows Test type.
Oracle Forms Extension	1.0.3	✓	✓	✗	✗	Deprecated in 10.5.1.

Terminal emulation

Supported software	Version	Functional Test perspective		Notes
		Record	Playback	
zSeries	3270	✔	✔	Supported via the terminal emulator application packaged with the product.
iSeries	5250	✔	✔	

Web browsers

Supported software	Version	Perspective				Notes
		Functional Test		UI Test		
		Record	Playback	Record	Playback	
Apple Safari	15	✘	✘	✔	✔	Supports the playback of mobile tests on iOS devices. Apple Safari support on desktop includes device modes.
	16	✘	✘	✔	✔	
Google Chrome	104 to 106	✔	✔	✔	✔	Supports the playback of mobile tests on Android devices. Deprecated in 10.5.3.
	107 to 114	✔	✔	✔	✔	Supports the playback of mobile tests on Android devices.

Supported software	Version	Perspective				Notes
		Functional Test		UI Test		
		Record	Playback	Record	Playback	
						Google Chrome browser support on desktop includes device modes.
Microsoft Edge	104 to 105	✓	✓	✓	✓	Deprecated in 10.5.3.
	106 to 113	✓	✓	✓	✓	
Microsoft Internet Explorer	11	✓	✓	✓	✓	Deprecated from 10.2.0.
Mozilla Firefox	103 to 105	✓	✓	✓	✓	Deprecated in 10.5.3.
	106 to 113	✓	✓	✓	✓	
Mozilla Firefox ESR	91	✓	✓	✓	✓	Supported inside the product container image.
	102	✓	✓	✓	✓	
Opera	89 and 90	✗	✗	✗	✓	Deprecated in 10.5.3.
	92 to 97	✗	✗	✗	✓	

Mobile applications

Apllication type	UI Test perspective		Notes
	Record	Playback	
Native Apps	✔	✔	
Hybrid Apps	✔	✔	
Web Apps	✘	✔	For Web Apps, recording is supported on desktop browsers in the device mode.

Supported software

The supported software section specifies the additional software that Rational® Functional TesterHCL OneTest™ UI supports.

Capabilities:

- Development tools on page 28
- Device clouds on page 29
- Integration Middleware on page 30

Development tools

Supported software	Version	Perspective		Notes
		Functional Test	UI Test	
Appium	1.21 and future fix packs	✘	✔	To execute Appium scripts as part of a compound test. Deprecated in 10.2.3.
	1.22 and future fix packs	✘	✔	To execute Appium scripts as part of a compound test.
Cucumber	Latest version	✔	✘	To integrate Cucumber features with test scripts.

Supported software	Version	Perspective		Notes
		Functional Test	UI Test	
eGit	5.13	✔	✔	Supports Eclipse source control.
IBM® Engineering Test Management (formerly known as IBM® Rational® Quality Manager)	7.0.1 and future fix packs	✔	✔	To execute test scripts from ETM.
	7.0.2 and future fix packs	✔	✔	
IBM® Engineering Workflow Management (formerly known as IBM® Rational® Team Concert™	7.0.1 and future fix packs	✔	✔	Supports Eclipse shell sharing.
	7.0.2 and future fix packs	✔	✔	
IBM® Rational® Performance TesterHCL OneTest™ Performance	10.5.3	✔	✔	Supports Eclipse shell sharing, facilitates Low Intensity Performance Testing and larger Accelerated Functional Testing.
IBM® Rational® Test WorkbenchHCL OneTest™ Studio	10.5.3	✔	✔	To integrate with Rational® Integration TesterHCL OneTest™ API and run tests.
Selenium	3	✘	✔	To execute Selenium scripts from a compound test.
	4	✘	✔	

Device clouds

Supported device clouds	Version	UI Test perspective	Notes
BitBar Cloud	Latest release	✔	To execute native, hybrid, and Web UI tests

Supported device clouds	Version	UI Test perspective	Notes
BrowserStack Cloud	Latest release	✔	on the mobile device cloud.
pCloudy Cloud	Latest release	✔	
Perfecto Mobile Cloud	Latest release	✔	

Integration Middleware

Supported software	Version	Perspective		Notes
		Functional Test	UI Test	
Apache Ant	1.9 and later	✔	✔	To initiate the test runs through Ant.
Azure DevOps	Latest release	✔	✔	To initiate the test runs from Azure.
HCL Launch	7.2.1.0	✔	✔	To initiate the test runs from HCL Launch.
	7.2.3.0	✔	✔	
	7.3	✔	✔	
Micro Focus ALM	12.6	✔	✔	
UrbanCode™ Deploy	7.2.1.0	✔	✔	To initiate the test runs from UrbanCode Deploy.
	7.2.3.0	✔	✔	
	7.3	✔	✔	
Jenkins	2.339	✔	✔	Deprecated in 10.5.3.
	2.346.2	✔	✔	
	2.361.1	✔	✔	
	2.375.2	✔	✔	To initiate the test runs from Jenkins.

Supported software	Version	Perspective		Notes
		Functional Test	UI Test	
	2.387	✔	✔	
	2.39	✔	✔	
Maven	3.5 and later	✔	✔	To initiate the test runs through Maven.
GitHub Actions	Latest	✔	✔	

Chapter 3. Getting Started Guide

This guide provides an overview of Rational® Functional TesterHCL OneTest™ UI. You can find the task flows to get you started with Rational® Functional TesterHCL OneTest™ UI. This guide is intended for new users.

Before you can perform the various tasks described in the *Getting Started Guide* and the other guides, you must install Rational® Functional TesterHCL OneTest™ UI. See Installation on page 201.

Before you start using Rational® Functional TesterHCL OneTest™ UI, see Rational Functional TesterHCL OneTest UI overview on page 32.

You can find information about getting started in the following perspectives of Rational® Functional TesterHCL OneTest™ UI:

- Getting started in the UI Test perspective on page 38
- Getting started in the Functional Test perspective on page 45

Rational® Functional TesterHCL OneTest™ UI overview

Rational® Functional TesterHCL OneTest™ UI is an object-oriented automated testing tool that tests a wide range of desktop, Web, and mobile applications. Rational® Functional TesterHCL OneTest™ UI provides automated testing capabilities for functional, regression, GUI, and data-driven testing.

Benefits of using Rational® Functional TesterHCL OneTest™ UI:

- Simplifies test visualization and editing using natural language and rendered smartshots.
- Enables testers to automate tests resilient to frequent application user interface changes with ScriptAssure technology.
- Lets you perform the same series of test actions with a varying set of test data.
- Combines a recorder of user actions with multiple customization options and intelligent script maintenance capabilities.
- Integrates with other solutions to provide access to work items and logical or compound SCM test asset support.
- Supports sharing of functional tests across team members and running on hybrid environments with integrations with Rational® Test Automation ServerHCL OneTest™ Server.

Rational® Functional TesterHCL OneTest™ UI runs on Windows® and Linux® platforms, while the web-based testing features run on Mac OS, in addition to Windows® and Linux®. For more information, refer to Operating systems on page 16.

Two perspectives: Functional Test and UI Test

Rational® Functional TesterHCL OneTest™ UI includes two perspectives that you can use to do your testing: the Functional Test perspective and the UI Test perspective.

The UI Test Perspective

You can use the UI Test perspective for Web, Mobile, and Windows tests. In addition, you can manage Selenium Java™ tests, Appium Java™ tests, and create compound tests with multiple test types, including functional tests, Web UI tests, mobile tests, Selenium and Appium tests.

Note: The capabilities provided in the UI test perspective are available in the Eclipse environment only.

For more information on the software that IBM® Rational® Functional TesterHCL OneTest™ UI supports to record and play back tests, see Recording and playback support on page 22.

Web UI Testing

IBM® Rational® Functional TesterHCL OneTest™ UI supports web applications built with HTML 4 and HTML 5 technologies on desktops and mobile devices. You can test web applications by using the industry-standard browsers such as Google Chrome, Apple Safari, Mozilla Firefox, and Microsoft Edge.

You can also test the web applications on dual monitors when you extend the display of your computer to a secondary monitor.

Mobile Testing

You can record and play back mobile tests on Android devices or emulators, iOS mobile devices or simulators. The UI Test perspective supports testing of native, hybrid, and web applications on both Android and iOS platforms. Specifically for web applications, you can record the tests in the device mode on a desktop browser and play back on the actual mobile devices.

You can also perform mobile application testing using the Appium test automation framework or in a device cloud environment. You can run JUnit Appium tests in an Appium framework.

Windows Testing

You can record and play back tests on the native Windows desktop applications. You can also test the Windows applications on dual monitors when you extend the display of your computer to a secondary monitor.

Rational® Functional TesterHCL OneTest™ UI supports the testing of the following applications:

- **Windows desktop applications**: The Windows desktop applications include two types of applications such as Universal Windows Platform (UWP) and Classic Windows applications.
- **Add-ins in MS Office**: The Microsoft Office applications such as Word, Outlook, Access, and PowerPoint are supported. Add-in applications which are integrated and available on the toolbar of the Office applications are also supported.

Distributed Testing

The Rational® Functional TesterHCL OneTest™ UI UI Test perspective helps you accelerate the test effort by providing ways to distribute test execution across multiple browsers and multiple computers simultaneously. Here are some of the capabilities provided by the UI Test perspective:

- The ability to run a single Web UI test on multiple browsers and mobile devices simultaneously
- The ability to run multiple Web UI tests on multiple browsers and mobile devices simultaneously
- The ability to run multiple Web UI tests across multiple remote computers simultaneously (Requires integration with Rational® Performance TesterHCL OneTest™ Performance)
- The ability to run multiple Web UI and compound tests simultaneously, both from the IDE and the command line
- The ability to run Web UI tests in the cloud
- The ability to record a Web UI test in one browser and play it back in another browser or on a mobile device
- The ability to test mobile Web applications on multiple mobile devices (Android and iOS) as a single test or as part of a compound test

The Functional Test Perspective

The Functional Test perspective in Rational® Functional TesterHCL OneTest™ UI is available in two integrated development environments (IDE) - Eclipse and Microsoft® Visual Studio .NET. Rational® Functional TesterHCL OneTest™ UI supports the following two scripting languages:

- The Java™ language in Eclipse integration.
- The VB.NET language and the Microsoft® Visual Studio .NET development environment in Microsoft Visual Studio .NET integration.

The Functional Test perspective can be used to test HTML4 applications, and native applications or hybrid applications that sometimes include embedded technologies.

Technology and Features

The object-oriented recording technology in the Rational® Functional TesterHCL OneTest™ UI Functional Test perspective lets you generate functional testing scripts for automated testing quickly by recording against the application under test. Rational® Functional TesterHCL OneTest™ UI uses object-oriented technology to identify controls or objects by their internal properties and not by screen coordinates. If the location or text of a control or the object changes, Rational® Functional TesterHCL OneTest™ UI can still find it during playback.

The object testing technology in Rational® Functional TesterHCL OneTest™ UI enables you to test any controls or object in the application under test, including the control properties and data.

Rational® Functional TesterHCL OneTest™ UI provides two distinct scripting styles - Java and Simplified scripting. The Java scripting style is based on Java programming, and the simplified scripting style is visual and natural language oriented. You can interact with the application visuals while viewing and editing the script.

While working with the Rational® Functional TesterHCL OneTest™ UI Eclipse Integration or Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, the test object maps are used and the application

visuals are not available. When you record a functional test script, Rational® Functional TesterHCL OneTest™ UI automatically creates a test object map for the application under test. The test object map lists the test objects available in the application, whether they are currently displayed or not. The object map provides a quick way to add objects to a functional test script. Since the test object map contains recognition properties for each object, you can easily update the recognition information in one central location. Any functional test scripts that use this test object map also share the updated information.

During recording you can insert verification points into the script to confirm the state of a control or an object across builds of the application under test. The verification point captures object information (based on the type of verification point) and stores it in a baseline data file. The information in this file becomes the baseline of the expected state of the object during subsequent builds. Rational® Functional TesterHCL OneTest™ UI has an object properties verification point and five data verification points (menu hierarchy, table, text, tree hierarchy, and list). You can use the Verification Point Comparator to analyze differences across builds and update the baseline file.

Rational® Functional TesterHCL OneTest™ UI features platform-independent and browser-independent test playback. For example, you can record a functional test script on Windows® and play it back on Linux®. You can record a functional test script using any of the browsers supported by the product and play back using any other supported browser. The functional testing script contains no references to the browser used during recording.

Proxy SDK

With Rational® Functional TesterHCL OneTest™ UI proxy software development kit (SDK) you can extend automated functional testing support for application user interface controls (GUI test objects), beyond what is provided by default.

Extension for Terminal-based Applications

Rational® Functional TesterHCL OneTest™ UI Extension for Terminal-based Applications tool helps you create test scripts to automate the functional testing of host application test cases. It provides a rich set of capabilities to test host attributes, host field attributes and screen flow. It uses terminal verification points and properties, as well as synchronization code to identify the readiness of terminal for user input.

Rational® Functional TesterHCL OneTest™ UI Extension for Terminal-based Applications supports functional testing of the applications listed in Terminal emulation on page 26.

Integrations

Both perspectives support integration with various products and that provides additional ways to run test scripts. The following integrations are supported:

- Development tools on page 28
- Device clouds on page 29
- Integration Middleware on page 30

Task flow for Setting up Rational® Functional TesterHCL OneTest™ UI

The diagram in this topic shows the task flow for setting up IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for functional testing using IBM® Rational® Functional TesterHCL OneTest™ UI.



Click a box for more information.
Shift-click to open a new browser.

Installation environment

Install a supported version of these items:

- Operating system
- Microsoft .NET Framework (for testing Windows)
- Microsoft Visual Studio .NET (for .NET scripting)

Install the product

Set up the test environment

Verify that a supported Java Runtime Environment (JRE) is available

Verify that a supported browser is available

Verify whether the environment for functional testing is enabled automatically

The test environment is not enabled

Manually enable the required environment for functional testing

Note: Check the compatibility between the operating systems, browsers, and JREs when you set up Rational® Functional TesterHCL OneTest™ UI.

Creating a project

The tests that you create, and the assets associated with the tests, reside in a project on your desktop. You can create the project separately, or you can simply record a test, which automatically creates a project named `testproj`.

1. Select **File > New > UI Test Project** or **Functional Test Project**.
   Result
   The **Create a Project** window opens.
2. In the **Project Name** field, enter a name for the project.
3. Select **Use default location**.
4. **Optional:** Click **Next** and select the folders to create in the new project.
   These folders organize your files by asset (`Tests`, `Results`, and so on).
5. Click **Finish**.
   Result

   After you click finish, you are prompted to record a test.

What to do next

You can create a test from a new recording or from an existing recording, or you can click **Cancel** to create a test project without recording a test.

Getting started in the UI Test perspective

With the UI Test perspective, you can find information and task flows to create and run functional tests for web applications.

You can find information to begin testing in the UI Test perspective as *Tutorials*. See Getting started with the Web UI Testing tutorial.

You can find the following task flows that you can use when you want to test in the UI Test perspective:

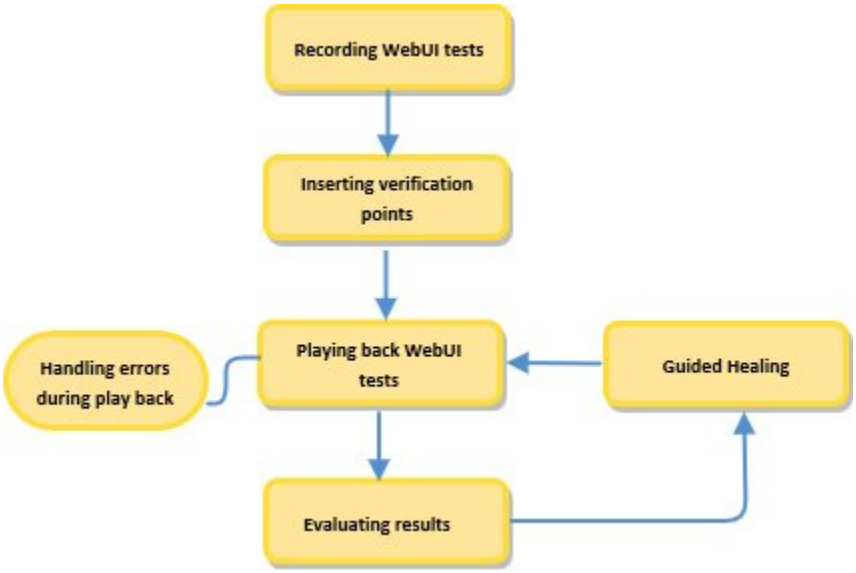
Task flow for testing web applications

The diagrams in this topic shows the standard task and advanced work flows for testing the web applications in IBM® Rational® Functional TesterHCL OneTest™ UI.

Task flow for standard Web UI test

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the standard task flow for testing web applications using IBM® Rational® Functional TesterHCL OneTest™ UI.

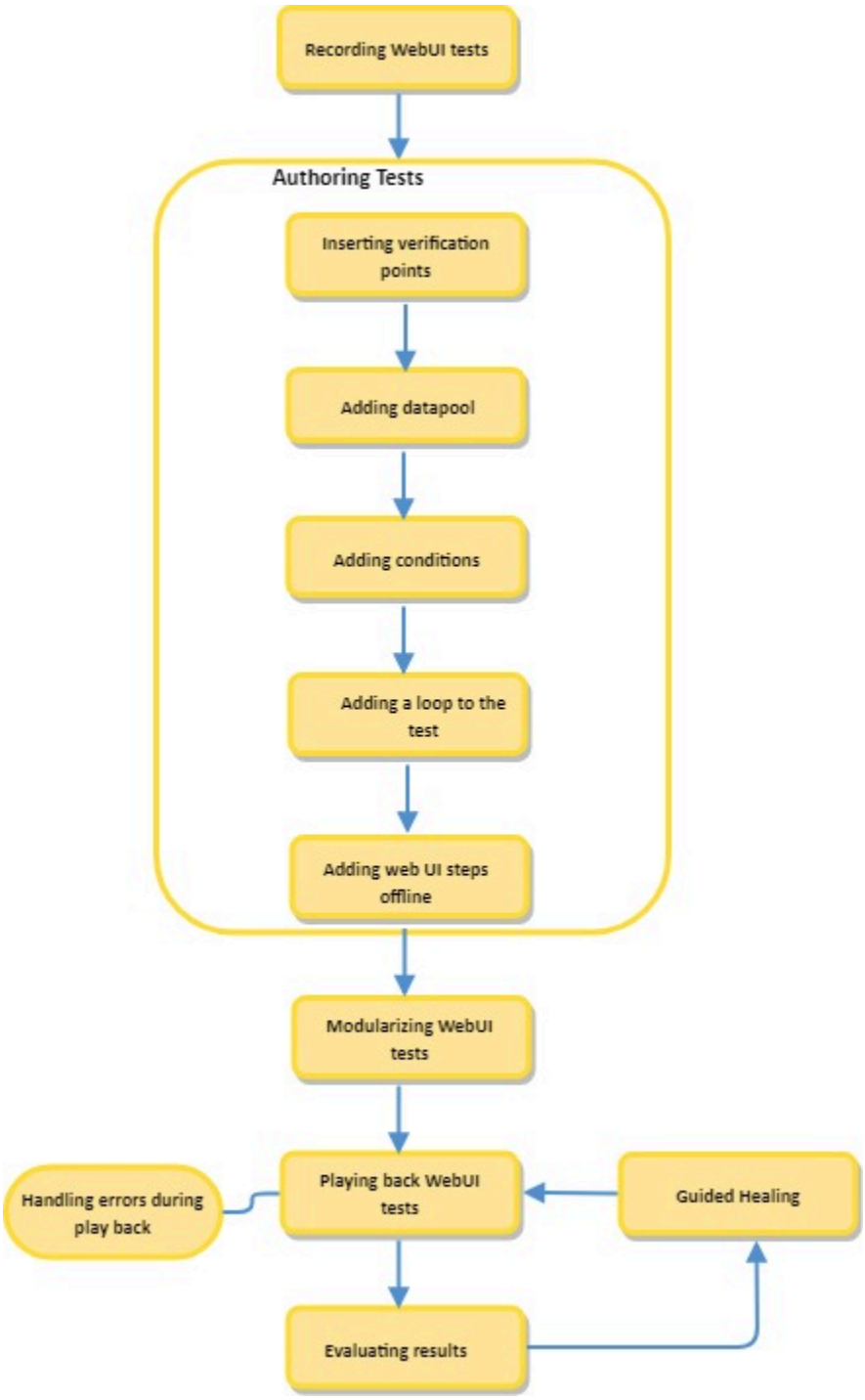


1. Click this area to get information about recording Web UI tests on page 532
2. Click this area to get information about inserting data verification points on page 907
3. Click this area to get information about playing back the Web UI tests on page 1235
4. Click this area to get information about evaluating results on page 1471
5. Click this area to get information about Guided Healing on page 618

Task flow for standard Web UI test

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the advanced task flow for testing web applications using IBM® Rational® Functional TesterHCL OneTest™ UI.



1. Click this area to get information about recording Web UI tests on page 532
2. Click this area to get information about inserting data verification points on page 907
3. Click this area to get more information about creating dataset on page 943
4. Click this area to get more information about adding conditions on page 589
5. Click this area to get more information about adding a loop to the test on page 587
6. Click this area to get more information about working on the test scripts on page 596
7. Click this area to get more information about modularizing tests on page 592
8. Click this area to get information about playing back the Web UI tests on page 1235
9. Click this area to get information about evaluating results on page 1471
10. Click this area to get information about Guided Healing on page 618

Task flows for testing mobile applications

You can find information about the tasks that you must perform to test Android mobile applications or iOS mobile applications.

Task flow for testing Android mobile applications

You must follow the tasks in this task flow to test Android mobile applications.

	Tasks	More information
1	Complete the prerequisite tasks before you record a test for Android mobile applications.	Prerequisite tasks for recording Android mobile applications on page 688
2	Configure an Android mobile application.	Configuring Android applications for mobile tests on page 690
3	Record a mobile test for Android mobile application.	Recording mobile tests for Android applications on page 692
4	Edit a mobile test.	Editing a mobile test on page 704
5.1	Run the recorded test for Android mobile applications from any of the following locations:	
	Computer that runs Rational® Functional TesterHCL OneTest™ UI.	Running mobile tests on an Android device or emulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1279
	Remote agent computer.	Running mobile tests on an Android device or emulator connected to a remote agent computer on page 1281
	BitBar Cloud.	Running mobile tests on Android devices on BitBar Cloud on page 1284
	BrowserStack Cloud.	Running mobile tests on Android devices on BrowserStack Cloud on page 1286
	Perfecto Mobile Cloud.	Running mobile tests on Android devices on Perfecto Cloud on page 1290
	pCloudy Cloud.	Playing back mobile tests on Android devices on pCloudy Cloud on page 1288
5.2	Alternatively, run the recorded test for Android mobile applications from the command line interface.	Running mobile tests for Android applications from the command-line on page 1415
6	View the test report.	Viewing unified reports on page 1455

Task flow for testing iOS mobile applications

You must follow the tasks in this task flow to test iOS mobile applications.

	Tasks	More information
1	Complete the prerequisite tasks before you record a test for iOS mobile applications.	Prerequisite tasks for recording iOS mobile applications on an iOS device on page 699
2	Configure an iOS application.	Configuring the iOS applications on page 700
3	Record a mobile test for iOS application.	Recording mobile tests for iOS applications on page 701
4	Edit a mobile test.	Editing a mobile test on page 704
5.1	Run the recorded test for iOS mobile applications from any of the following locations:	
	Computer that runs Rational® Functional TesterHCL OneTest™ UI.	Running mobile tests on an iOS device or simulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1311
	Remote agent computer.	Running mobile tests on an iOS device or simulator connected to a remote computer on page 1313
	BitBar Cloud.	Running mobile tests on iOS devices on BitBar Cloud on page 1315
	BrowserStack Cloud.	Running mobile tests on iOS devices on BrowserStack Cloud on page 1317
	Perfecto Mobile Cloud.	Running mobile tests on iOS devices on Perfecto Cloud on page 1321
	pCloudy Cloud.	Running mobile tests on iOS devices on pCloudy Cloud on page 1319
5.2	Alternatively, run the recorded test for iOS mobile applications from the command line interface.	Running mobile tests for iOS applications from the command-line on page 1424
6	View the test report.	Viewing unified reports on page 1455

Task flows for testing Windows desktop applications

You can find information about the tasks that you must perform to test Windows desktop applications.

You must follow the tasks in this task flow to test Windows desktop applications.

	Tasks	More information
1	Configure a Windows desktop application.	Configuring applications for tests on page 536

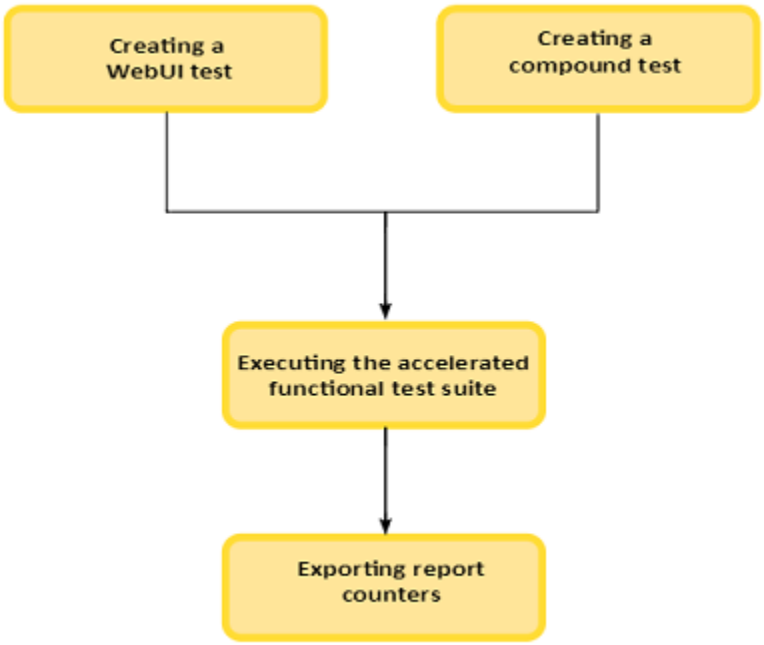
	Tasks	More information
2	Record a Windows test for the configured application.	Recording a Windows test on page 709
3	Edit a Windows test.	Editing a Windows test on page 713
4	Play back a Windows test.	Running a Windows test on page 1351
5	View the test report.	Unified reports on page 1455

Task flow for standard accelerated functional testing

The diagram in this topic shows the standard task flow for the accelerated functional testing for Web UI applications using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the standard task flow for accelerated functional testing using IBM® Rational® Functional TesterHCL OneTest™ UI



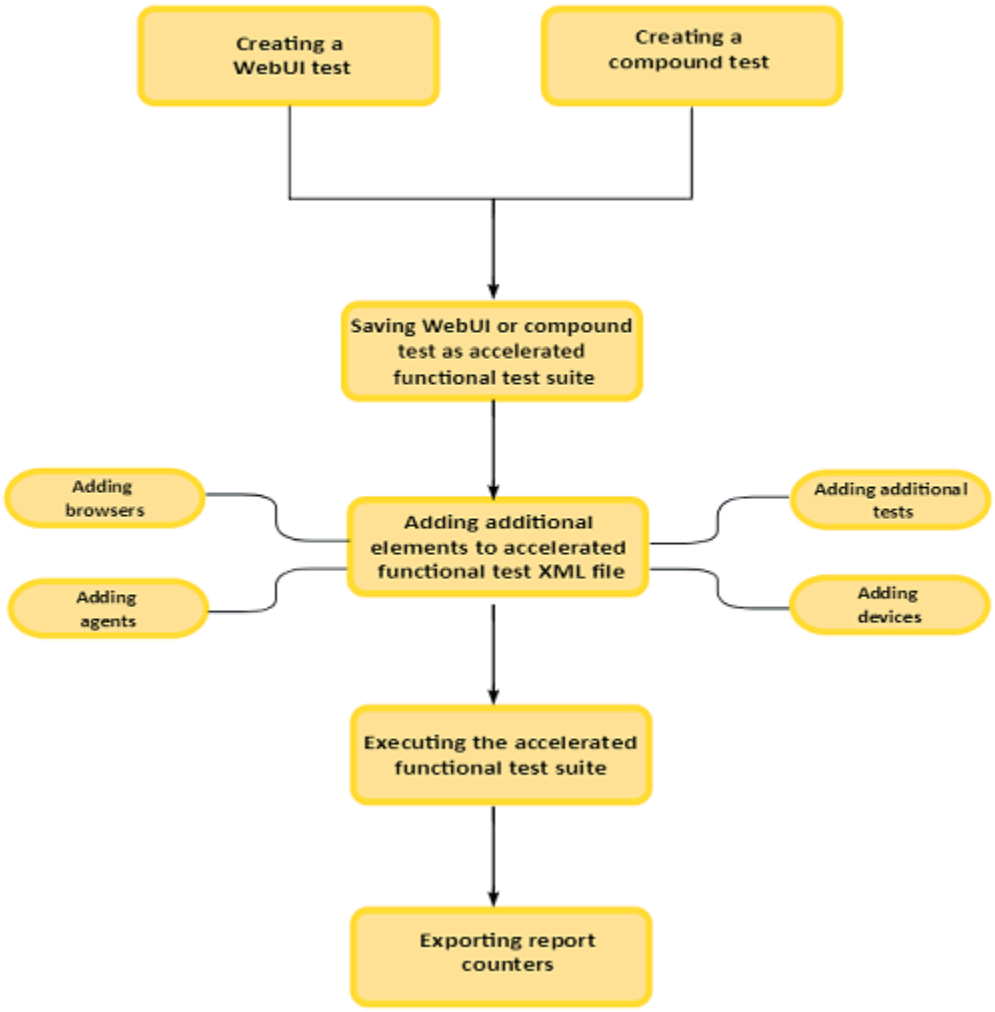
1. Click this area to get information about recording Web UI tests on page 532
2. Creating a compound test on page 738
3. Configuration of test runs from the command line on page 1381

Task flow for advanced accelerated functional testing

The diagram in this topic shows the advanced task flow for the accelerated functional testing for Web UI applications using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the advanced task flow for accelerated functional testing using IBM® Rational® Functional TesterHCL OneTest™ UI



1. Click this area to get information about recording Web UI tests on page 532
2. Creating a compound test on page 738
3. 4 on page 746
4. Creating an Accelerated Functional Test asset on page 746
5. Configuration of test runs from the command line on page 1381

Getting started in the Functional Test perspective

To get started in functional testing, familiarize yourself with some basic concepts and use cases. You can consider several approaches before you start creating the functional test scripts to effectively automate test cases.

You can find information to begin testing in the Functional Test perspective as *Tutorials*. See Tutorial for getting started with Functional Testing.

Planning functional tests

You must consider your requirements and identify the right approach to follow to automate functional testing. By automating functional testing, you can effectively use the features available in Rational® Functional TesterHCL OneTest™ UI.

- **Modular functional test scripts**: You can create multiple functional test scripts for a test case and reuse the scripts that apply to different test cases. By creating modular functional test scripts, you can reuse the test scripts for different test cases. By using modular test scripts, you can also associate the test scripts with multiple test cases that are created in a test management tool such as IBM® Rational® Quality Manager. In Rational® Functional TesterHCL OneTest™ UI, you can use the Call Script option to call another functional test script.
- **Recording test scripts versus manual test scripts**: In Rational® Functional TesterHCL OneTest™ UI, you can either record a functional test script against the test application or manually create a test script. If the application under test changes frequently and impacts the test-object hierarchy, you must update the test object maps to avoid errors or failures. The test object maps are created when you record the functional test scripts against the test application. In this scenario, you can create manual test scripts and use the find() methods to search for the test objects in the application and specify the action to use on them. For manual test script creation, you must be familiar with the programming language.
- **Simplified test scripts versus Java test scripts**: Simplified scripts are functional test scripts in the form of simple English statements that are easy to understand and edit. With the simplified script feature enabled, you can also view the corresponding Java test script in the Java script editor. When you edit the simplified script, the Java script reflects the changes in the Java script editor. You can switch to the Java script editor using the Insert Java Code Snippet or Insert Java Method features available in the simplified script editor and start working with the Java test script.

Information resources

To get started with IBM® Rational® Functional TesterHCL OneTest™ UI, there are various resources that will help you to understand the process of functional testing and the features that you can use for effectively automating the test cases.

Introduction

Rational Functional TesterHCL OneTest UI overview on page 32

List of domains/technologies that are supported for functional testing on page 1703

Interactive workflows

Task flow for testing applications on page 47

Task flow for testing Java applications on page 49

Task flow for testing HTML applications on page 51

Task flow for testing Eclipse applications on page 53

Task flow for testing SAP applications on page 55

Task flow for testing Flex applications on page 57

Tutorials

Get started with functional testing using simplified scripts on page 90

Create a functional test using Java scripts on page 104

Perform a data-driven functional test using Java scripts on page 124

Test GEF applications on page 138

Test Adobe Flex application on page 134

Extend Rational Functional TesterHCL OneTest UI capabilities using Proxy SDK on page 142

Tutorials for testing in the Visual Studio IDE on page 147

Samples

Sample project to test a Java application on page 180

Functional testing proxy SDK samples

Troubleshooting

Troubleshooting on page 1507

The Functional Test perspective

The Functional Test perspective has components that are displayed automatically when you start Rational® Functional TesterHCL OneTest™ UI: the product menu, toolbar, Projects view, Script (simplified) editor, Java™ editor, Script Explorer, Console view, Tasks view, Application view, Keyword view, Properties view, and the status bar.

Click a component for more information.

- Menu on page 1762
- Toolbar on page 1777
- Projects view on page 860
- Script editor on page 1889
- Java Editor on page 1785
- Script Explorer on page 1813
- Application View on page 1888
- Keyword View on page 500

- Properties view: Properties view - General page on page 1890, Properties View- Playback page on page 1892, Properties View - Log page on page 1892 and Properties View - Advanced page.
- Console View on page 1744
- Tasks View on page 1838
- Status bar -- the product uses the status bar at the bottom of the Test Perspective to display messages.

For more information on perspectives and views, see the online *Workbench User Guide*.

Importing a sample functional test project

One way to get started quickly with Rational® Functional TesterHCL OneTest™ UI is to import the sample functional test project that is provided with IBM® Rational® Functional TesterHCL OneTest™ UI. The sample project includes sample test scripts that you can use. When you import the sample project, it is copied to your Eclipse workspace.

1. Start Rational® Functional TesterHCL OneTest™ UI and click **File > Import**.
2. In the Import wizard, expand **Functional Test**, select **Sample Functional Test project**, and click **Next**.
3. In the **Review project content** page, view the project and scripts that are going to be imported and click **Finish**.
   Result

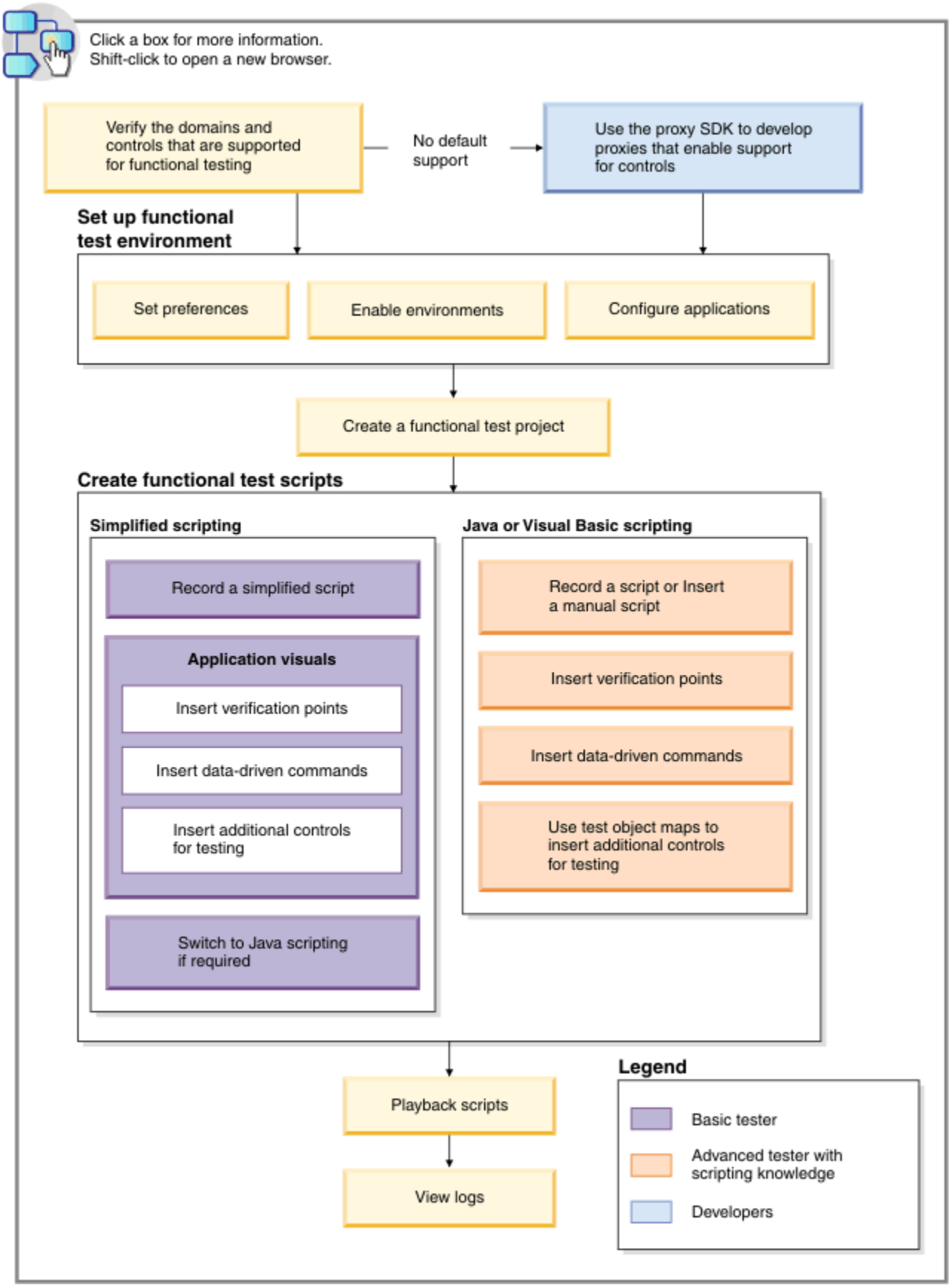
   A sample Functional Test project named **FunctionalTesterDemo** opens in the Project Explorer.
4. To re-import the sample project, repeat Step 1 and Step 2 You will be prompted whether to overwrite the sample project in your workspace or to create a new sample project.

Task flow for testing applications

The diagram in this topic shows the task flow for testing the application using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for functional testing using IBM® Rational® Functional TesterHCL OneTest™ UI.



1. Click this area to get information about the supported domains and the controls for functional testing on page 1703
2. Click this area to get information about the proxy SDK on page 1078

Note: Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

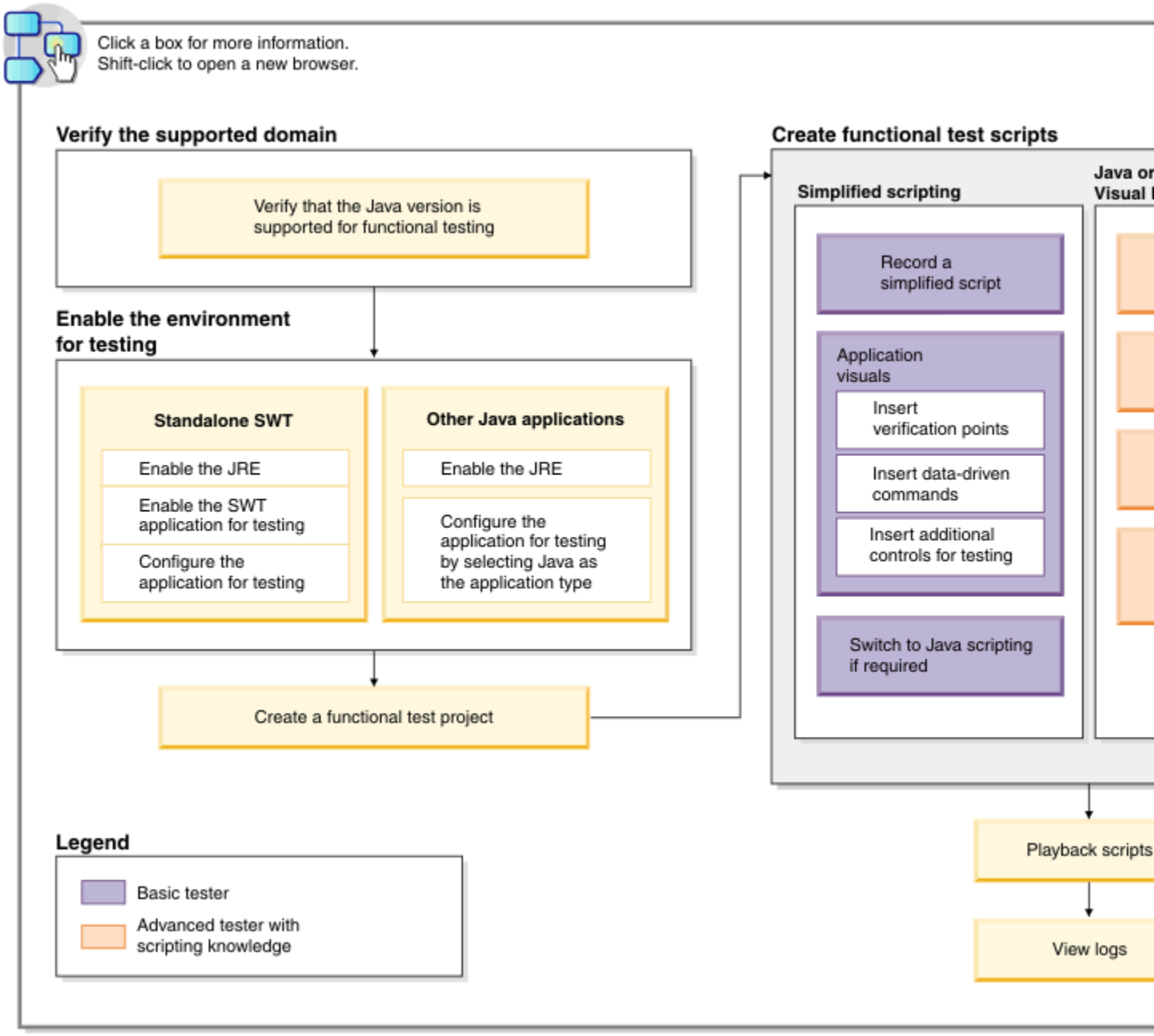
Developers: A developer who knows theRational® Functional TesterHCL OneTest™ UI framework and programming based on the proxy framework can write proxies for adding support to test the controls that are not supported by default for functional testing.

Task flow for testing Java applications

The diagram in this topic shows the task flow for testing the Java application using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for testing Java applications using IBM® Rational® Functional TesterHCL OneTest™ UI.



1. Click this area to get information about the supported Java versions on page 1716
2. Click this area to get information about enabling the JRE on page 778
3. Click this area to get information about enabling the SWT application on page 795
4. Click this area to get information about configuring the application for testing on page 796
5. Click this area to get information about enabling the JRE on page 778
6. Click this area to get information about configuring the application for testing on page 796
7. Click this area to get information about creating a functional test project on page 858
8. Click this area to get information about recording a simplified script on page 866
9. Click this area to get information about inserting verification points using the application visuals on page 934
10. Click this area to get information about inserting a data-driven commands into a scripts by using an application visual on page 944
11. Click this area to get information about inserting an application control into the script by using an application visual on page 874

Note:

Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

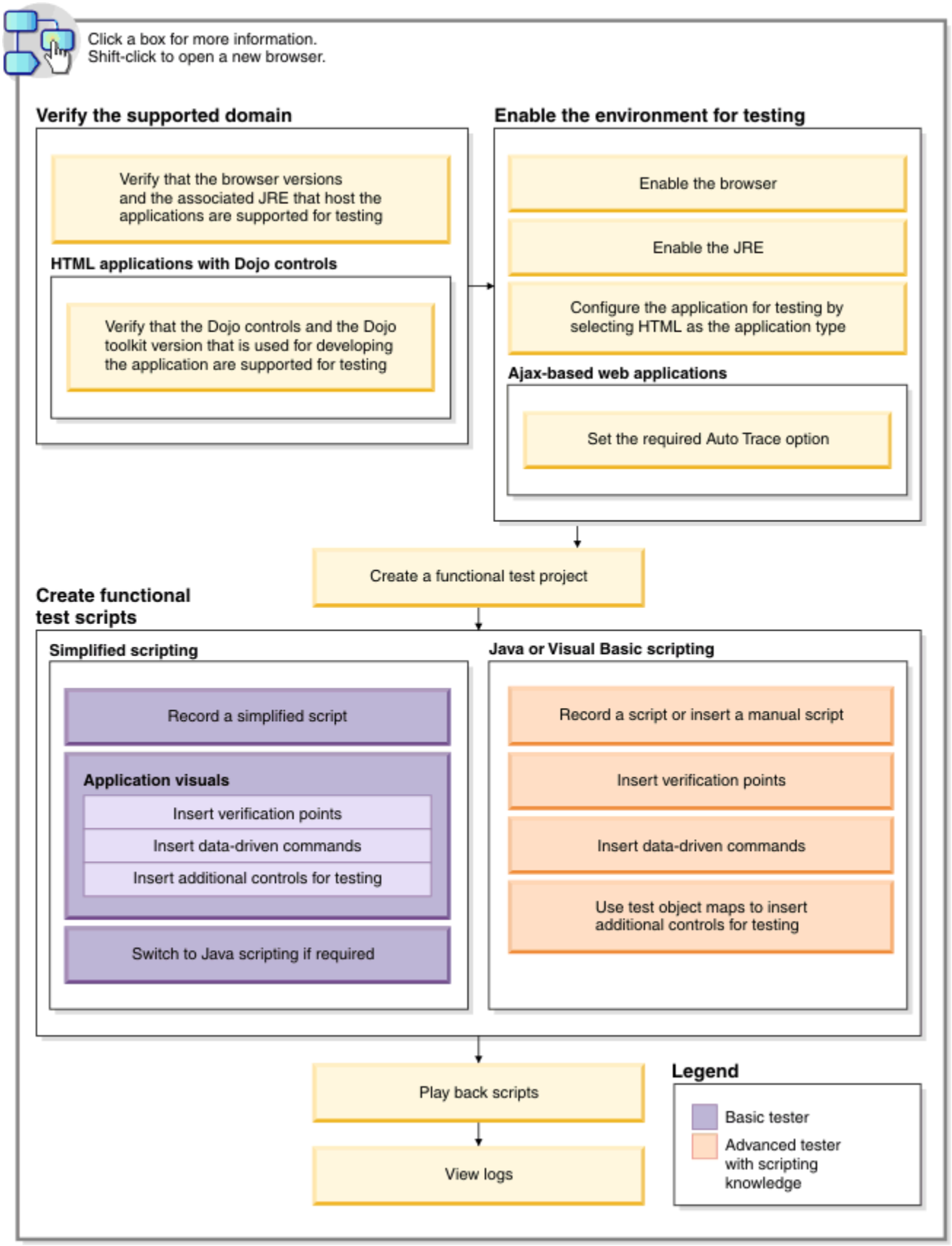
Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

Task flow for testing HTML applications

The diagram in this topic shows the task flow for testing the HTML application using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for testing HTML applications using IBM® Rational® Functional TesterHCL OneTest™ UI.



1. Click this area to get information about the supported browsers and their versions on page 1715
2. Click this area to get information about the supported Dojo controls for functional testing on page 1709
3. Click this area to get information about enabling the browser on page 780
4. Click this area to get information about enabling the JRE on page 778

Note: Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

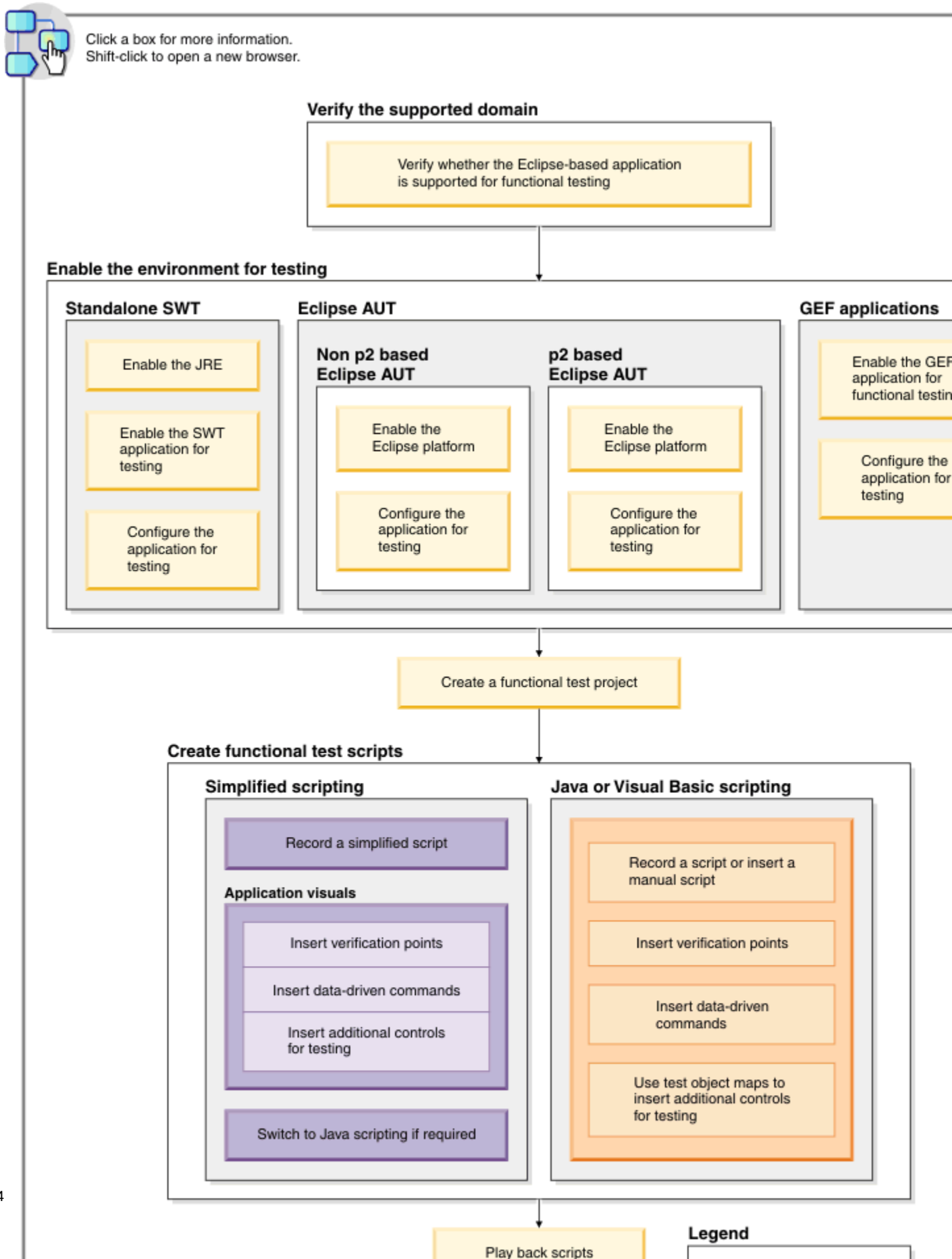
Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

Task flow for testing Eclipse applications

The diagram in this topic shows the task flow for testing the Eclipse application using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for testing Eclipse applications using IBM® Rational® Functional TesterHCL OneTest™ UI.


Click a box for more information.
Shift-click to open a new browser.
Verify the supported domain
Verify whether the Eclipse-based application is supported for functional testing
Enable the environment for testing
Standalone SWT
Enable the JRE
Enable the SWT application for testing
Configure the application for testing
Eclipse AUT
Non p2 based Eclipse AUT
Enable the Eclipse platform
Configure the application for testing
p2 based Eclipse AUT
Enable the Eclipse platform
Configure the application for testing
GEF applications
Enable the GEF application for functional testin
Configure the application for testing
Create a functional test project
Create functional test scripts
Simplified scripting
Record a simplified script
Application visuals
Insert verification points
Insert data-driven commands
Insert additional controls for testing
Switch to Java scripting if required
Java or Visual Basic scripting
Record a script or insert a manual script
Insert verification points
Insert data-driven commands
Use test object maps to insert additional controls for testing
Play back scripts
Legend

Note: Rational® Functional TesterHCL OneTest™ UIautomatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing.

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

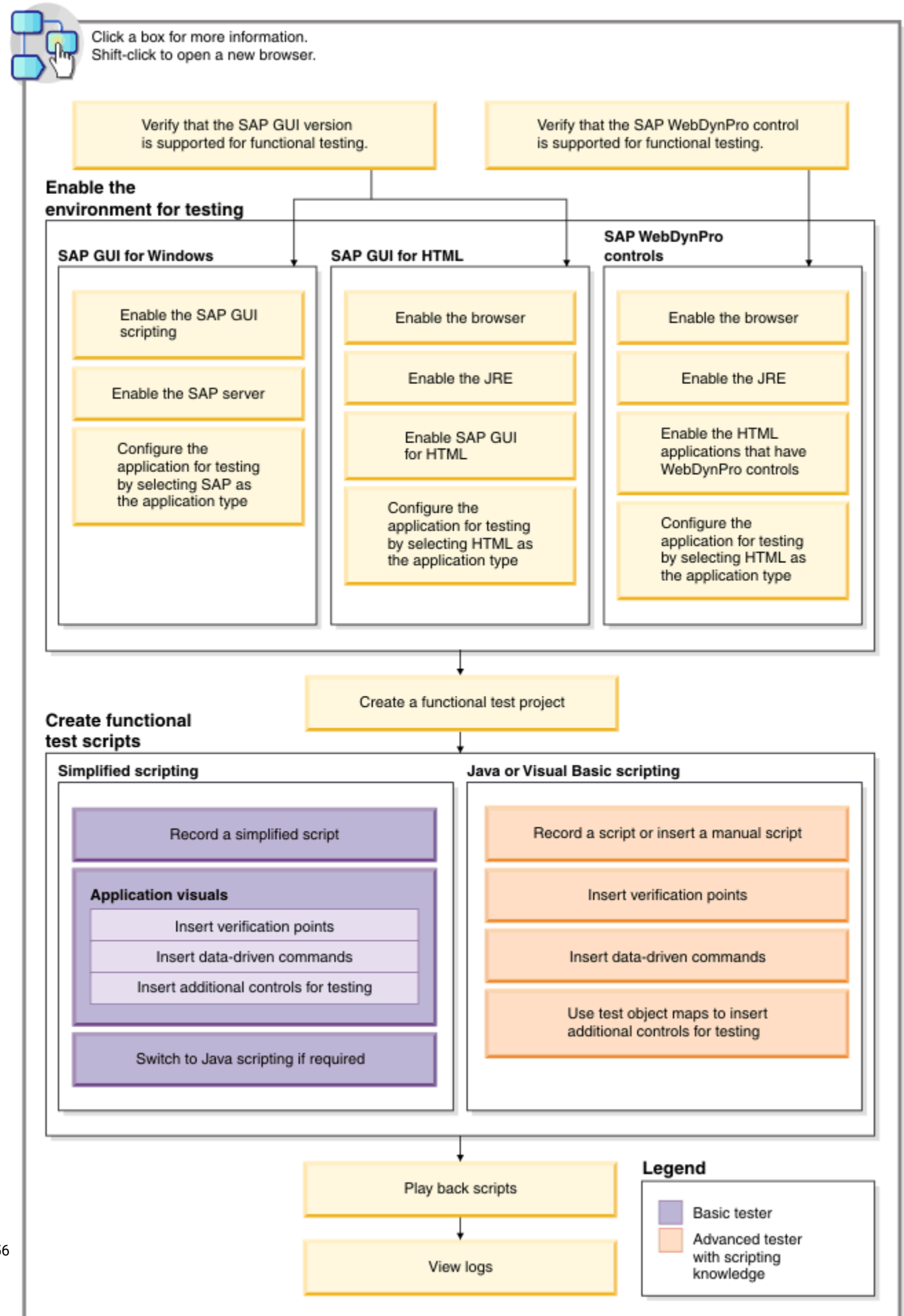
Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

Task flow for testing SAP applications

The diagram in this topic shows the task flow for testing the SAP application using IBM® Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for testing Java applications using IBM® Rational® Functional TesterHCL OneTest™ UI.


Click a box for more information.
Shift-click to open a new browser.
Verify that the SAP GUI version is supported for functional testing.
Verify that the SAP WebDynPro control is supported for functional testing.
Enable the environment for testing
SAP GUI for Windows
Enable the SAP GUI scripting
Enable the SAP server
Configure the application for testing by selecting SAP as the application type
SAP GUI for HTML
Enable the browser
Enable the JRE
Enable SAP GUI for HTML
Configure the application for testing by selecting HTML as the application type
SAP WebDynPro controls
Enable the browser
Enable the JRE
Enable the HTML applications that have WebDynPro controls
Configure the application for testing by selecting HTML as the application type
Create a functional test project
Create functional test scripts
Simplified scripting
Record a simplified script
Application visuals
Insert verification points
Insert data-driven commands
Insert additional controls for testing
Switch to Java scripting if required
Java or Visual Basic scripting
Record a script or insert a manual script
Insert verification points
Insert data-driven commands
Use test object maps to insert additional controls for testing
Play back scripts
View logs
Legend
Basic tester
Advanced tester with scripting knowledge

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

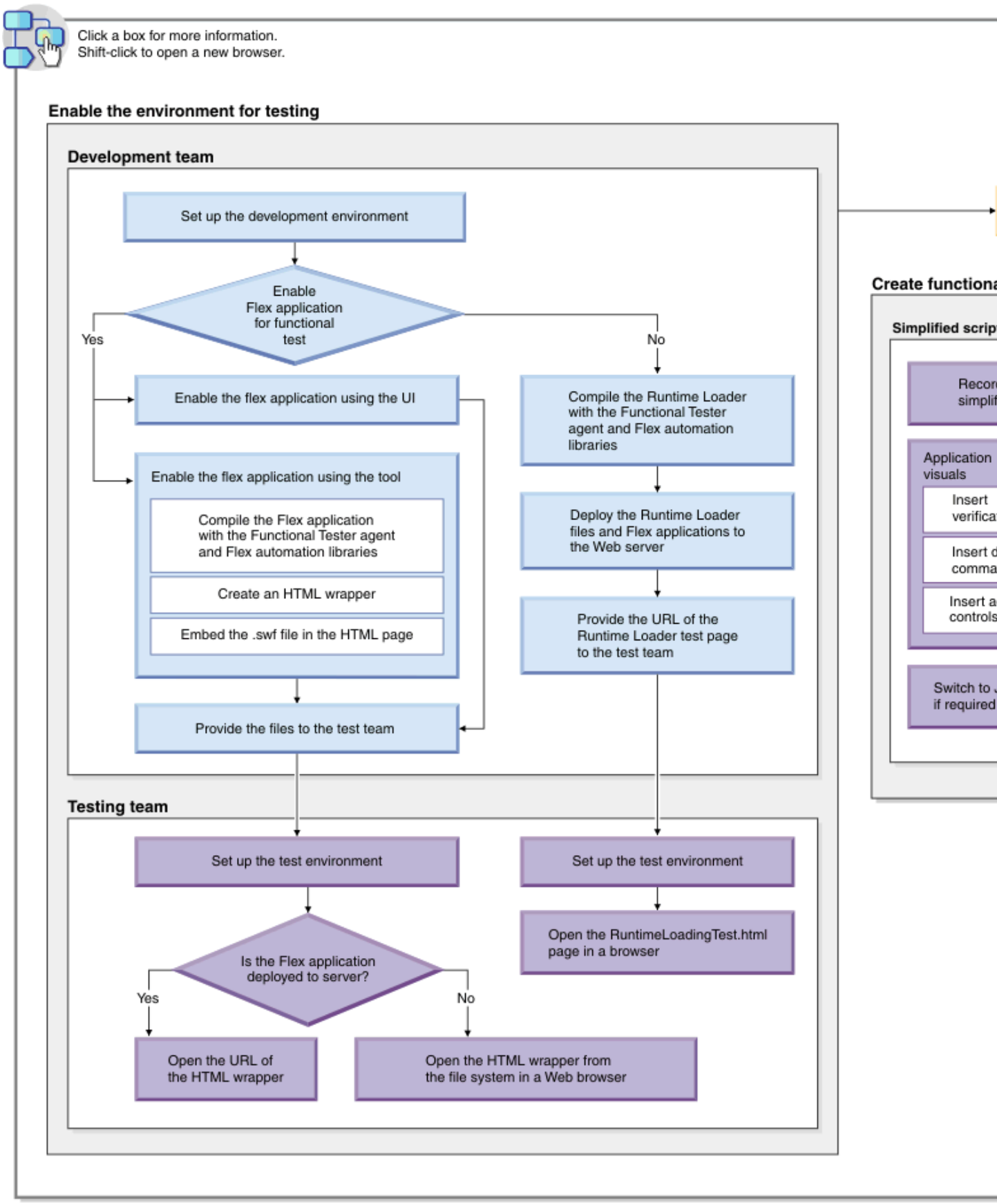
Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

Task flow for testing Flex applications

The diagram in this topic shows the task flow for testing the Flex application using Rational® Functional TesterHCL OneTest™ UI.

Although the diagram implies that each task is completed sequentially, you can do many of the tasks at the same time.

The diagram shows the task flow for testing Java applications using Rational® Functional TesterHCL OneTest™ UI. For information about the Flex testing process, see Flex applications testing process on page 809



1. Click this area to get information about setting up the development environment for testing Flex applications on page 811
2. Click this area to get information about enabling a Flex application for testing on page 809

Basic tester: A basic tester can record functional test scripts that are generated as simplified test scripts. The tester does not require programming knowledge to edit the functional test scripts. The tester can switch to Java scripting, and use the Insert Java Code Snippet or Insert Java Method features that are available in the simplified script editor. The tester then starts to work with the Java test script directly. You can use the application visuals to insert verification points, data-driven commands, and additional controls for testing.

Advanced tester with scripting knowledge: A tester with Java or Visual Basic programming knowledge can either record functional test scripts or create the test scripts manually. You can use the test object maps to update the objects and insert additional objects for testing.

Developers: A developers enables the Flex applications for testing and provides it to the testers for functional testing.

Visual Studio .NET window

The Visual Studio .NET window has the following components: Start page, product menu, toolbar, Script Explorer, Solution Explorer, code and text editor, Server Explorer, Class view, Properties window, and Dynamic Help.

Start Page , which is displayed each time you start the integrated development environment (IDE). The Start Page contains three tabs: Projects (default tab), Online Resources, and My Profile. You can use the Projects tab to create a new project or open an existing project. The Online Resources tab provides the latest developer news, helps you create new solutions, and promotes interaction with the online developer community. The My Profile tab enables you to personalize settings for the keyboard layout, Visual Studio .NET window layout, and online Help. For more information, see the Visual Basic .NET online help.

After you open a project and select a script, the Visual Studio .NET window contains the following components:

Rational® Functional TesterHCL OneTest™ UI menu on page 1902

Rational® Functional TesterHCL OneTest™ UI toolbar on page 1910

Functional Test Script Explorer on page 1813

Solution Explorer, which provides an organized view of your projects and their files and ready access to the commands that pertain to them. By default, the Solution Explorer displays on the right side of the Visual Studio .NET window.

Note: If you want to simplify the Visual Studio .NET user interface, you can click the **My Profile** tab on the **Start** page, select **Student Window Layout** in the Window Layout list, and select **(no filter)** in the Help Filter list. The Solution Explorer and Dynamic Help display on the left side and the Script Explorer displays on the right side of the Visual Studio .NET window. To return to the default Visual Studio .NET user interface, click

the **My Profile** tab on the **Start** page, and select **Visual Studio Developer** in the Profile list. In the product menu, click **View > Script Explorer**.

Code and Text Editor, which is a word-processing utility. It is referred to as either the Text Editor or the Code Editor, based on its content. For more information, see the Visual Basic .NET online help.

Server Explorer, which Rational® Functional TesterHCL OneTest™ UI does not use.

Class View, which is a programmatic view of symbols in your code, for example, namespaces, classes, methods, and functions. Symbols are organized by projects. Within each project, the contents are shown in a hierarchical tree view that indicates the containment relationships between the symbols. Other structural information, such as base classes and interfaces and overrideable methods are also included. You can use this logical view of your solution and its projects to better understand the structure of your code and the relationships of its various symbols. For more information, see the Visual Basic .NET online help.

Properties Window, which you can use to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the properties window to edit and file, project, and solution properties. For more information, see the Visual Basic .NET online help.

Dynamic Help, which provides pointers to information specific to the current area you are using or task you are trying to complete within the IDE. For more information, see the Visual Basic .NET online help.

Using the Functional Test perspective of on Linux

For the Functional Test perspective, most features are supported on Linux®, except for recording the scripts. This topic provides an overview about how Rational® Functional TesterHCL OneTest™ UI behaves on Linux.

Testing applications on Linux

Test scripts

All of the functionality of the product works on Linux except for the recorder. You can test the applications on Linux® in two different ways.

- Record functional test script on Windows® and export it to Linux®. You can then play back on Linux®.
- You can launch the browser on Linux by adding the step `startBrowser("Chrome", <url>);` or `startBrowser("Firefox", <url>);` in the test script.

  Note:
  - Do not add or enable the browser using the options available in the **Enable Environments** wizard.
  - To launch the browser, use `StartBrowser()` instead of `StartApp()`.

- Ensure that you pass the fully qualified URL (with the protocol) as a parameter to `startBrowser()`. For example, http://www.ibm.com/.

- You can write the test scripts using the Test Object Insert tool. Instead of creating an object map through recording, populate it outside by opening the map and select objects in the test application. For information, see Creating a New Test Object Map.

  See the API Reference Help and the Advanced Topics on page 1179 for examples of scripting to solve certain problems.

Verification points

You can also insert verification points without recording, by opening the Verification Point wizard from the Script Explorer.

Launcher scripts

Rational® Functional TesterHCL OneTest™ UI provides the following scripts that can be executed from the command-line:

- To start Rational® Functional TesterHCL OneTest™ UI: `<installation directory>/ft_starter`
- To set the test environment variable: `source <installation directory>/FunctionalTester/bin/rtsetup`

dataset functionality

The datasets functionality works on Linux®, but there may be slight differences.

Java™ and HTML support

On Linux®, you can only test Java™ and HTML applications. Some Linux® GUI applications, like those developed with Motiff, are not supported.

Installing the product on Linux®

When you install the Rational® Functional TesterHCL OneTest™ UI package, the only feature you can install on Linux® is Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration. Before you install the product on Linux, make sure that you have the following libraries on your machine.

- libXm.so.4
- libstdc++.so.5
- libXp.so.6
- gtk2.i686
- gtk2-engines.i686
- PackageKit-gtk-module.i686
- PackageKit-gtk-module.x86_64
- libcanberra-gtk2.x86_64
- libcanberra-gtk2.i686

Note: For more information about the library dependencies on Linux, refer to https://www.ibm.com/developerworks/community/blogs/qualitymanagement/entry/rft_libraries_dependency_on_linux?lang=en.

Starting the test application outside the script

If you start your test application on Linux® outside Rational® Functional TesterHCL OneTest™ UI (not using a startApp command or other script call), you must first set the environment variables. You must make sure that the `LD_PRELOAD` and `FT_INSTALL_DIRECTORY` are properly set in the environment from which they start the test application, else Rational® Functional TesterHCL OneTest™ UI will not be able to properly play back scripts against the application.

Set up the following variables depending upon the shell that you are using.

```
export FUNCTIONAL_TESTER_DIR= <FT installation directory> For e.g, /opt/caspian
```

```
export FT_CUSTOMIZATION_DIRECTORY=$FUNCTIONAL_TESTER_DIR/bin/customization
```

```
export FT_CONFIGURATION_DIRECTORY=$FUNCTIONAL_TESTER_DIR/bin/configuration
```

```
export FT_INSTALL_DIR=$FUNCTIONAL_TESTER_DIR/bin
```

```
export FT_ECLIPSE_DIR=$FUNCTIONAL_TESTER_DIR/eclipse
```

```
export FT_JRE=$FUNCTIONAL_TESTER_DIR/eclipse/jre
```

```
export LD_PRELOAD=$FUNCTIONAL_TESTER_DIR/bin/libftevent.so
```

For Linux platforms that are based on GTK + 2.18 and higher, set`GDK_NATIVE_WINDOWS=true`

For HTML based test applications, the LD_LIBRARY_PATH must be set to the browser installation directory.

Note: Alternatively, you can use `rtsetup` script to set all the functional tester environment variables. Refer to the Launcher scripts section.

Enabling JREs on Linux®

To run the enabler for enabling the JREs on Linux®, you must have root privileges.

Functionality not supported on Linux

Recorder

You cannot record on Linux®. You can record scripts on Windows® and then play them back on Linux®.

Tutorials

The tutorials provided with Rational® Functional TesterHCL OneTest™ UI involve recording scripts. To learn the product functionality, do the tutorials on the Windows® platform. You can play back the scripts you recorded using the tutorial on Linux®.

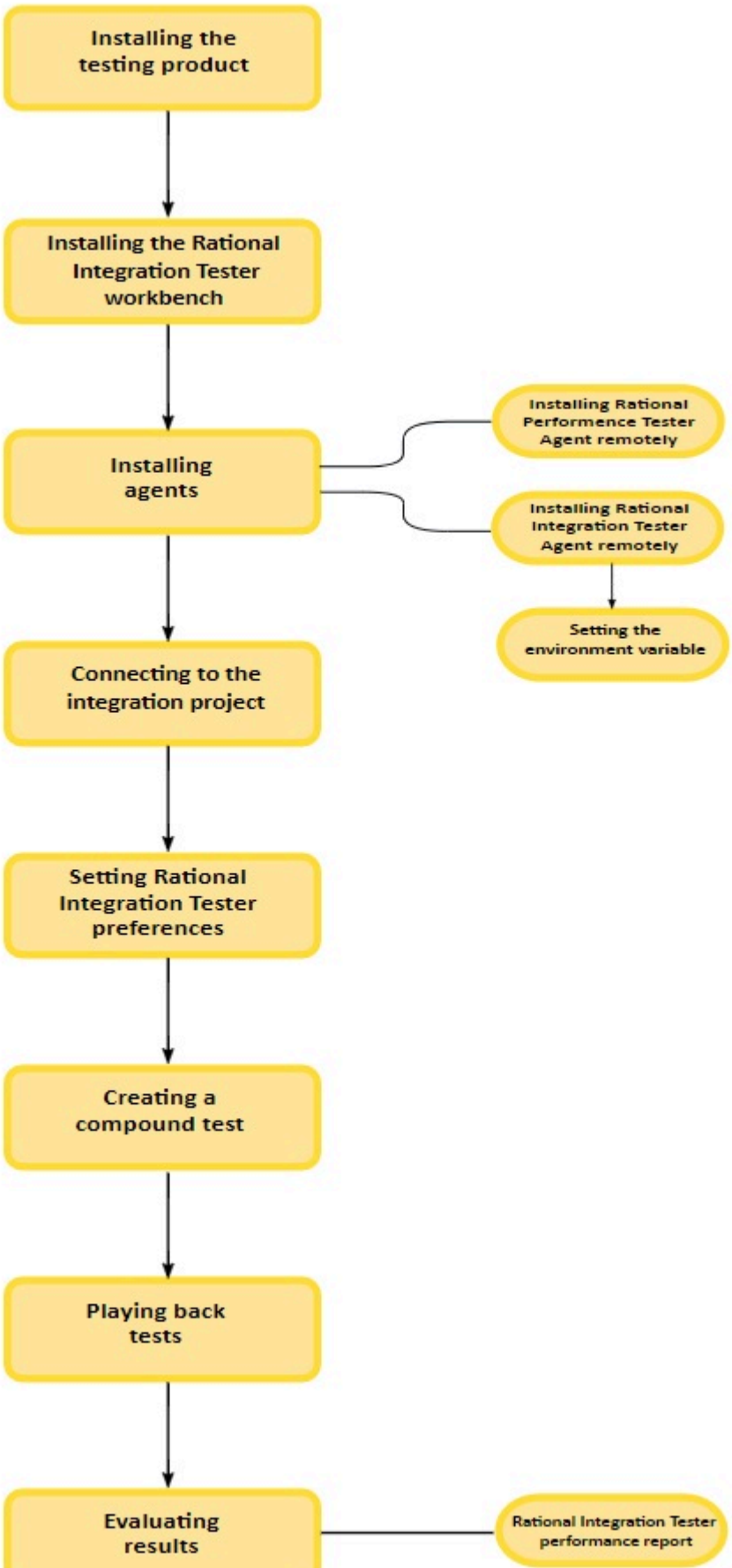
Task flow for integrating Rational® Functional TesterHCL OneTest™ UI and Rational® Integration TesterHCL OneTest™ API

You can run integration tests in IBM® Rational® Functional TesterHCL OneTest™ UI by using IBM® Rational® Functional TesterHCL OneTest™ UI Extension for Rational® Integration TesterHCL OneTest™ API. In IBM® Rational® Functional TesterHCL OneTest™ UI, you can create a compound test to run the integration tests by using agents.

To integrate tests, you must install Rational® Functional TesterHCL OneTest™ UI Extension for Rational® Integration TesterHCL OneTest™ API. Also, to execute the tests remotely, you must install Rational® Performance TesterHCL OneTest™ Performance Agent and Rational® Integration TesterHCL OneTest™ API Agent.

After you installed the required software, you must set the environment variable and connect to the integration project. To open the Rational® Integration TesterHCL OneTest™ API project from Rational® Functional TesterHCL OneTest™ UI Test Navigator, you must set the path to the execution file in the **HOT-API IntegrationRIT**

Integration preferences. Later, you must create a compound test and play back the test to evaluate the



Chapter 4. Tutorials

You can learn to start using Rational® Functional TesterHCL OneTest™ UI by reviewing the information in the Tutorials for the UI Test perspective and Functional Test perspective.

You can find the following tutorials:

- Tutorials for testing in the UI Test perspective on page 65.
- Tutorials for testing in the Functional Test perspective on page 90.

Tutorials for testing in the UI Test perspective

You can use the tutorials to get started with testing in the UI Test perspective in Rational® Functional TesterHCL OneTest™ UI.

Prerequisites

Before you can get started with testing by using the tutorials, you must have completed the following tasks:

- Installed Rational® Functional TesterHCL OneTest™ UI. See Installation on page 201.
- Verified the system requirements specified for Rational® Functional TesterHCL OneTest™ UI. See System Requirements on page 13.

Be sure to start Rational® Functional TesterHCL OneTest™ UI on a new workspace.

Testing in the UI Test perspective

In the UI Test perspective, you record tests from web applications. The UI Test perspective provides you a recording and natural language scripting environment to test browser applications that use HTML5 and JQuery. You can test on Internet Explorer, Mozilla Firefox, Google Chrome, or Safari browsers.

A web application includes many user scenarios. For example, on an e-commerce website many users only browse the products, some of the users sign in and add products to the cart, and a few of them purchase the products and sign out of the web site.

In this tutorial, you can learn to record actions on the web application, create verification points, run the test, view the test results, split actions in the test, create a dataset and associate it with the test, create a compound test, and to add a loop to the test. You can use *https://en.wikipedia.org* as the application under test (AUT).

You can find the following tutorials that you can use to get started with testing in the UI Test perspective:

- Lesson 1: Recording actions on an AUT on page 66
- Lesson 2: Creating a verification point on page 68
- Lesson 3: Running a test on page 70
- Lesson 4: Test log and reports on page 72
- Lesson 5: Splitting actions in a test on page 76

- Lesson 6: Creating a dataset on page 81
- Lesson 7: Associating a dataset with a test on page 82
- Lesson 8: Creating a compound test on page 86
- Lesson 9: Adding a loop on page 88

Disclaimers

All the screen captures that are used in the lessons of this tutorial pertain to Rational® Functional TesterHCL OneTest™ UI version 10.5.

Lesson 1: Recording actions on an AUT

You can learn about recording actions that you perform on a web application in the UI Test perspective.

Before you begin

You must have completed the following tasks:

- Created valid credentials to log in to Wikipedia.

  Note: You can create the credentials from the Create account page.

- Verified that the JavaScript is enabled to record and play back the UI tests in a browser.
- Installed the Google Chrome extension. See Enabling Google Chrome for Web UI testing on page 540.

About this task

Rational® Functional TesterHCL OneTest™ UI automatically records all actions that you perform on your web application for a specific test scenario.

Note: You must wait for each page to load completely when you record the test on your web application.

1. Click **File > New > Test From Recording** in the UI Test perspective.
   Result
   The **Recording Session** dialog is displayed.
2. Perform the following steps in the **Recording Session** dialog:
   a. Click **Create a test from a new recording**.
   b. Select **Web UI Test**, and then click **Next**.
   Result
   The **Select Location** dialog is displayed.
3. Click the **Create the parent folder** icon .
   Result
   The **Create a UI Test project** dialog is displayed.
4. Perform the following steps in the **Create a UI Test project** dialog:

a. Enter *myProj* in the **Project name** field.
b. Select the **Use default location** checkbox, if it is not selected, and then click **Finish**.

Result

The **Create a UI Test project** dialog is closed.

5. Enter *wiki-search* in the **Test name** field of the **Select Location** dialog, and then click **Next**.

   Result

   The **Select Client Application** dialog is displayed.
6. Select **Google Chrome** as the client application for recording, and then click **Next**.

   Result

   The **Google Chrome Recorder Settings** dialog is displayed.
7. Accept the default settings, and then click **Finish**.

   Result

   The recording is started and the Google Chrome webpage is displayed.
8. Enter `https://en.wikipedia.org` in the address bar.

   Result

   The home page of the Wikipedia website is displayed, which is the application under test (AUT).
9. Perform the following actions in the AUT that you want to record:

   a. Click **Log in**.

      Result

      The Wikipedia login page is displayed.

   b. Enter your Wikipedia user credentials, and then click **Log in**.

      Result

      The home page of the Wikipedia website is displayed with your username.

   c. Search for the name of an animal in the **Search Wikipedia** field.

      For example, search for *Elephant*.

      Result

      The Wikipedia article of the animal is displayed.

   d. Click the **Talk** tab.

      Result

      The talk page for the article is displayed.

   e. Click **Log out**.

      Result

      The Wikipedia logout page is displayed and you are successfully logged out.

10. Click the **Stop** icon  in the **Recording Control** view of Rational® Functional TesterHCL OneTest™ UI to stop the recording that you performed.

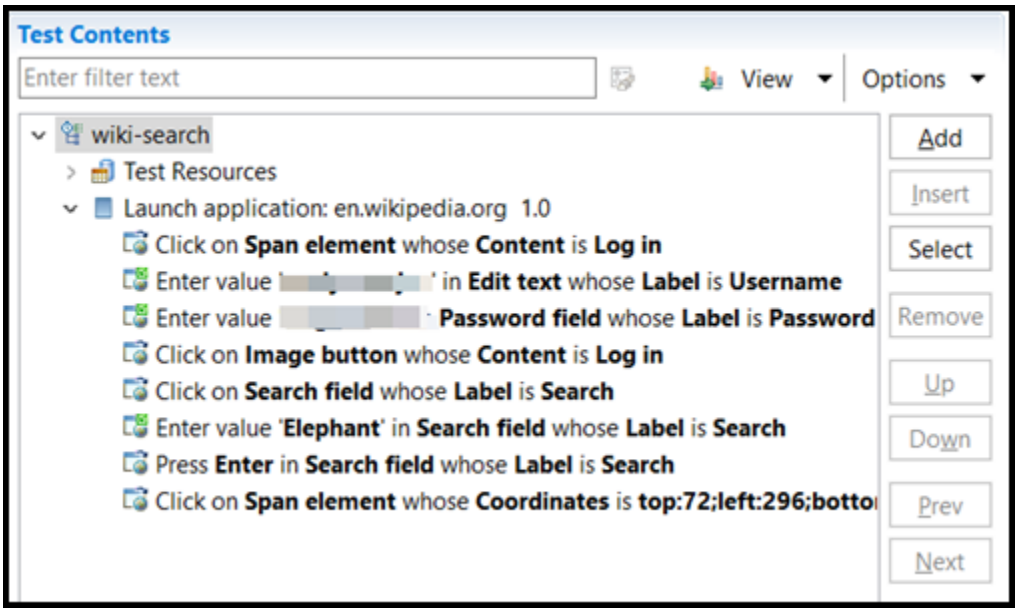
    Result

    A message is displayed that the test generation is complete.
11. Perform the following steps in the **Test Navigator** panel:

a. Click **myProj > Tests**.

b. Double-click **wiki-search**.

The generated test is displayed in the **Test Contents** panel of the test editor.



Results

In this lesson, you have learned to record actions that you performed on a web application in the UI Test perspective.

Lesson 2: Creating a verification point

You can learn about creating a verification point for an action that you recorded on an application under test (AUT). Verification points verify that a certain action has taken place, or they verify the state of a control or an object during the play back of tests.

Before you begin

You must have completed the following tasks:

- Generated a Web UI test. See Lesson 1: Recording actions on an AUT on page 66.
- Read and been familiar with verification points. See Verification points on page 904.
- Ensured that the **Test Contents** panel of the test editor window is displayed.

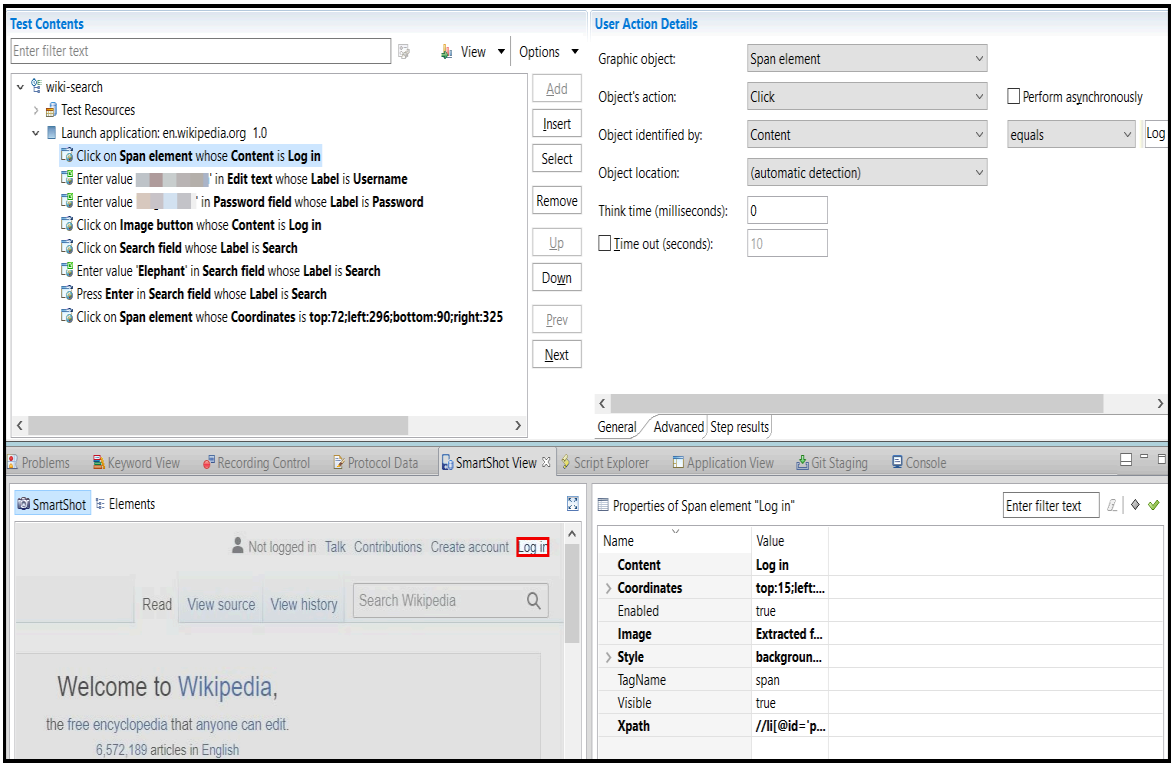
About this task

You can create a verification point for a recorded action in the **SmartShot View** tab of the test editor window and view the verification step in the test.

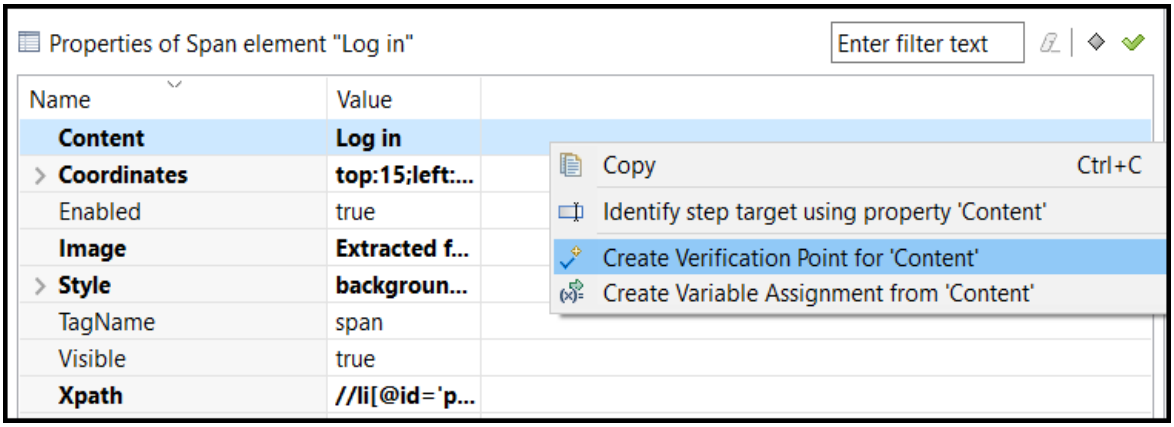
1. Select the test step that corresponds to the first click action that you performed on the **Log in** link on the Wikipedia website.

   The screen capture and properties for the click action are displayed in the **SmartShot View** tab.

   Note: Verify whether the click action is highlighted in the screen capture that is displayed in the **SmartShot** subtab.

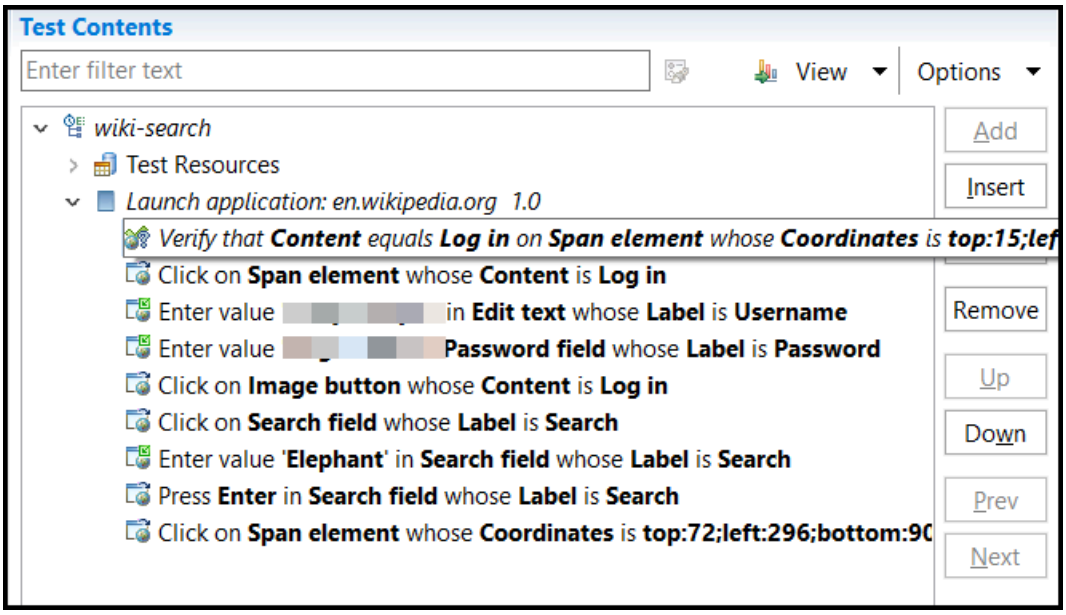


2. Right-click the row with **Log in** in the **Properties** panel of the **SmartShot View** tab.



3. Select **Create verification point for 'Content'**.

A verification point is created for the selected click action and the verification step is added in the **Test Contents** panel with the icon.



4. Click **File > Save** to save the changes.

Results

In this lesson, you have learned to create a verification point for an action that you recorded on an AUT.

What to do next

You can learn about running a test to verify whether the test scenario was played back successfully by Rational® Functional TesterHCL OneTest™ UI. See Lesson 3: Running a test on page 70.

Lesson 3: Running a test

You can learn about running a test to verify whether a test scenario is played back successfully by Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Generated a Web UI test. See Lesson 1: Recording actions on an AUT on page 66.
- Created a verification point for an action that you recorded in the test. See Lesson 2: Creating a verification point on page 68.
- Read and been familiar with **Think time**. See Editing Web UI tests on page 573.
- Ensured that the **Test Contents** panel of the test editor window is displayed.

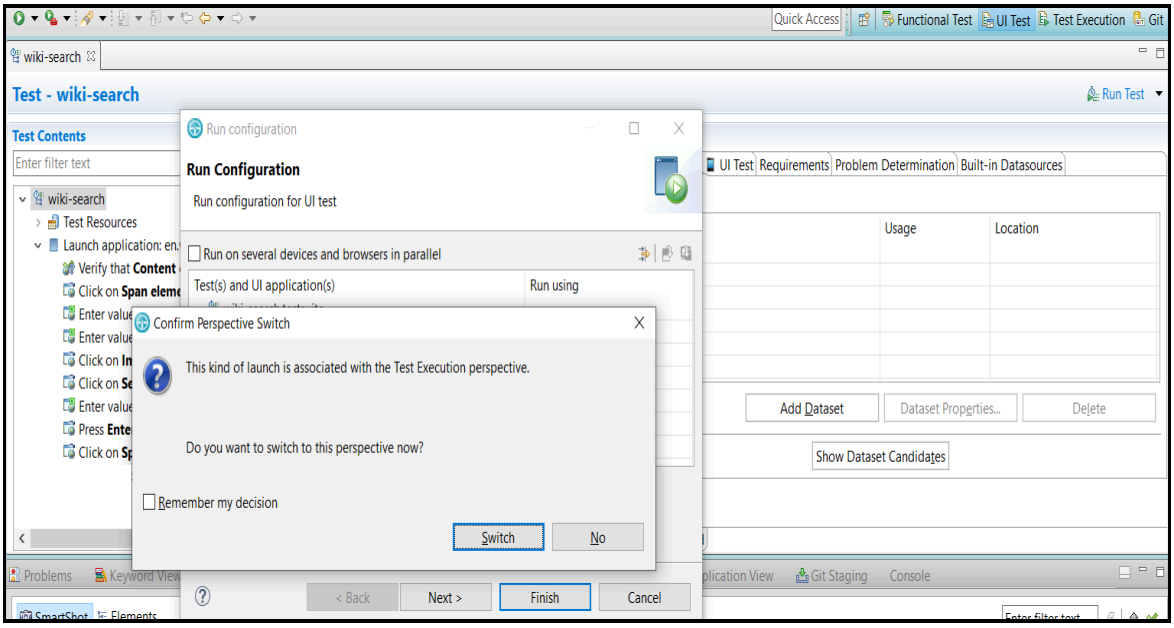
About this task

For the test scenario in this tutorial, you can run the recorded test on the Google Chrome browser.

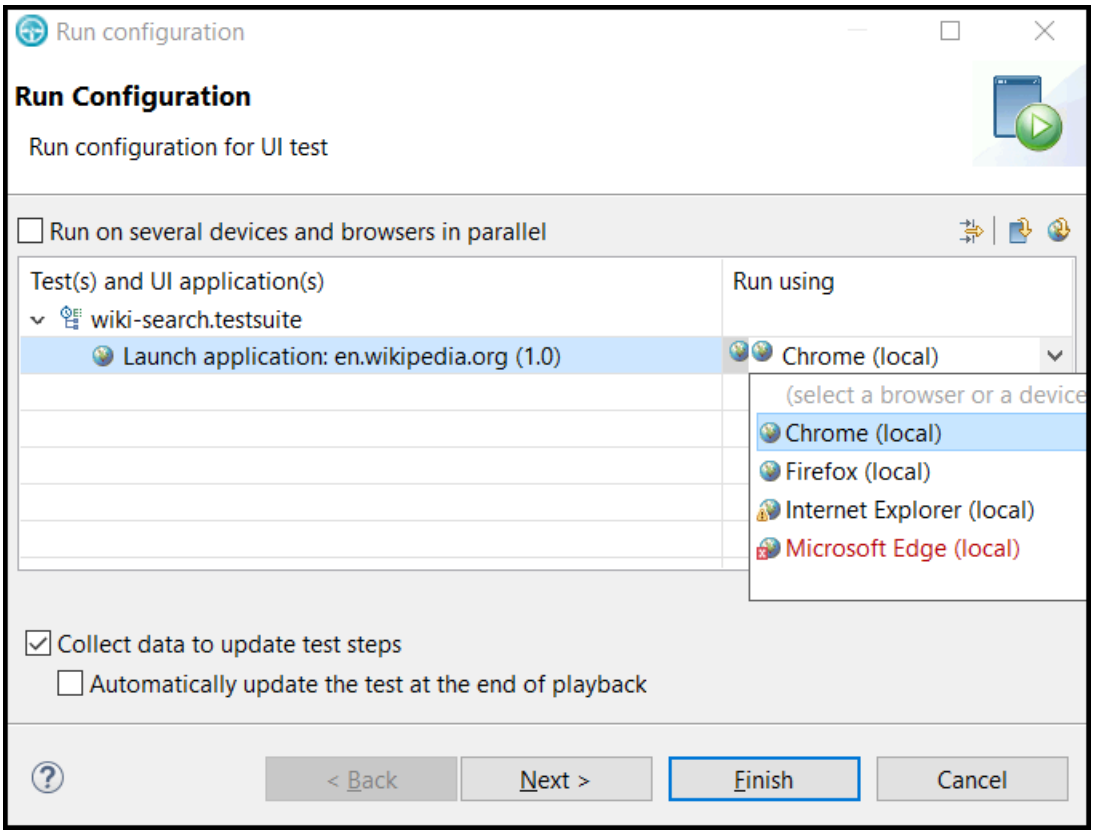
Note: Before the test run, you can remove any waiting time (**Think time**) that you spent on a control or a test line.

1. Click **Run Test** in the test editor window of the **UI Test** perspective.

   The **Confirm Perspective Switch** dialog is displayed.



2. Click **Switch** to run the test in the **Test Execution** perspective.
3. Select **Chrome** in the **Run Configuration** dialog.



4. Click **Finish**.

   The test run begins.

Results

You have learned to run a test.

At the end of the test run, Rational® Functional TesterHCL OneTest™ UI generates the test log and reports.

What to do next

You can learn about the test log and reports that are generated by Rational® Functional TesterHCL OneTest™ UI. See Lesson 4: Test log and reports on page 72.

Lesson 4: Test log and reports

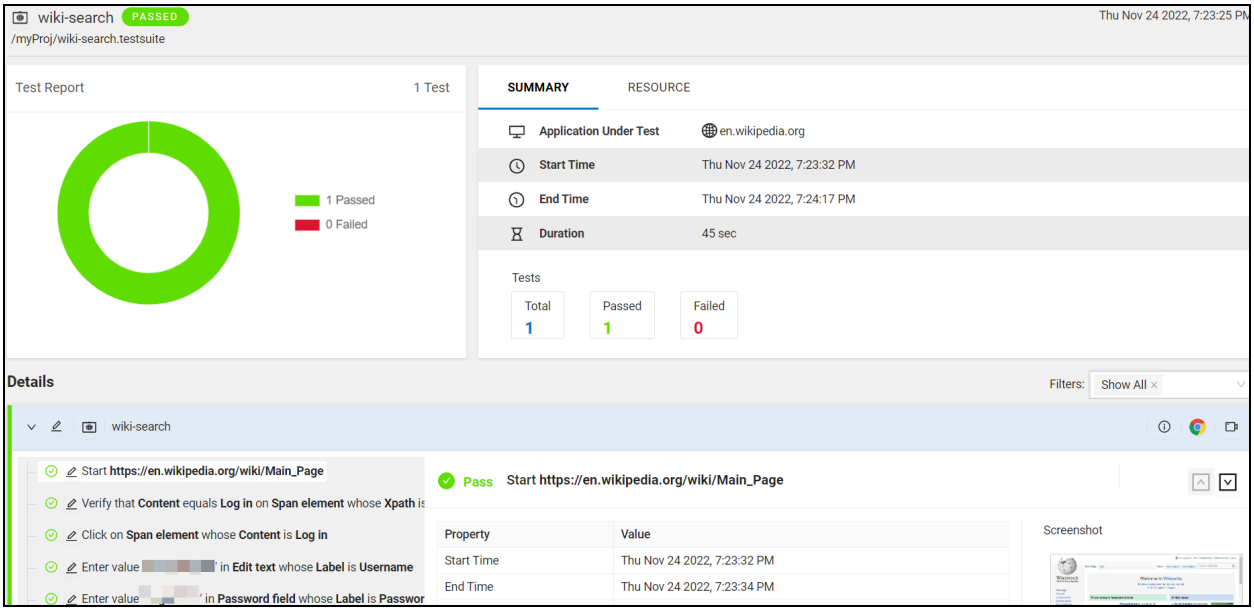
You can learn about the test log and reports that are generated by Rational® Functional TesterHCL OneTest™ UI after a test run is complete. You can use the data in the log and reports to analyze the test results and confirm whether the test run was successful.

Rational® Functional TesterHCL OneTest™ UI generates the unified report, UI Test Statistical report, and Test Log after a test run is complete.

Unified reports

Unified reports provide an overview of the test results. Rational® Functional TesterHCL OneTest™ UI generates the unified reports immediately after a test run. If the test run fails, an error message is displayed against the failed step in the report. See Unified reports on page 1455.

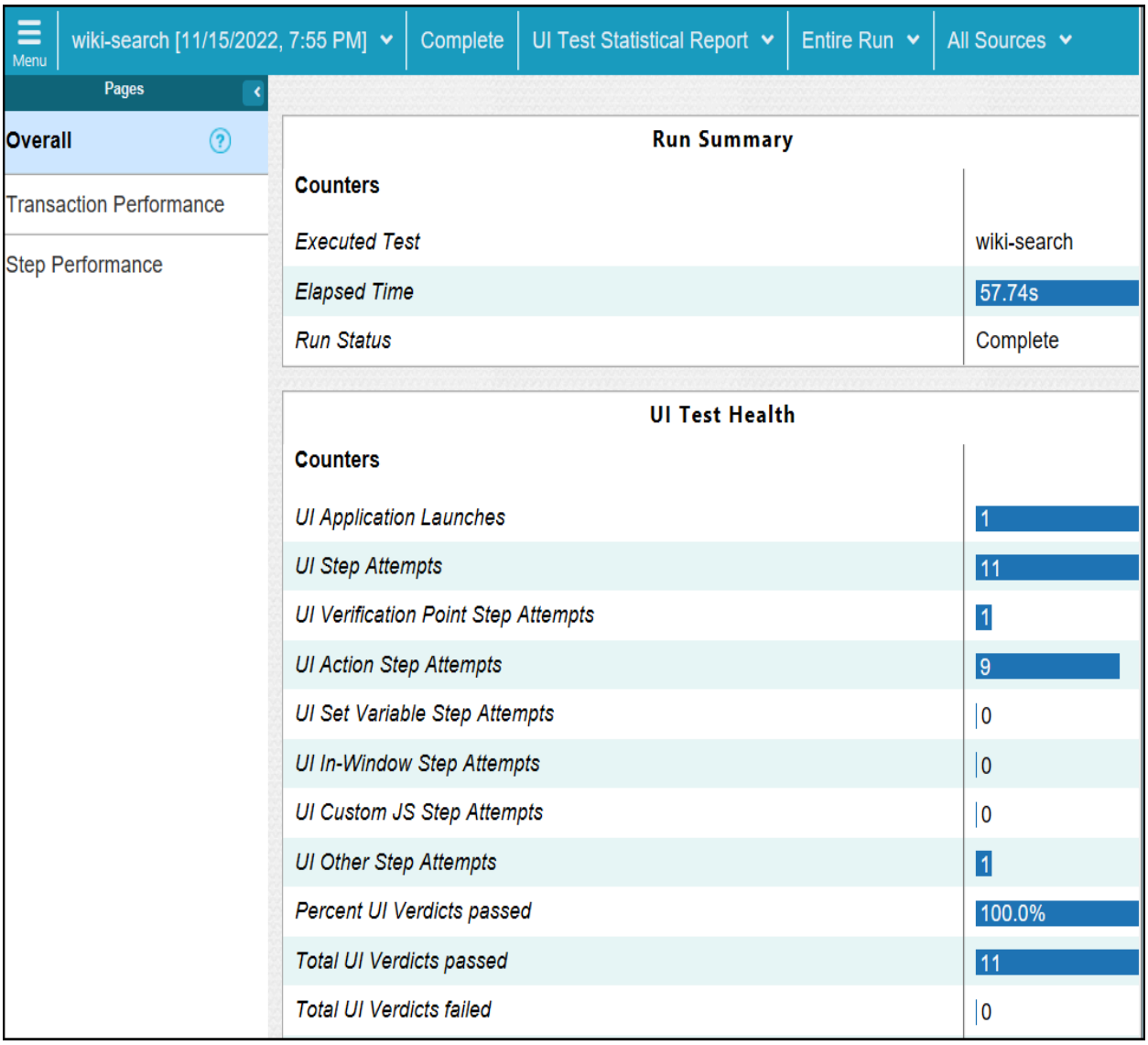
The following screen capture displays the unified report that is generated for the test scenario of this tutorial:

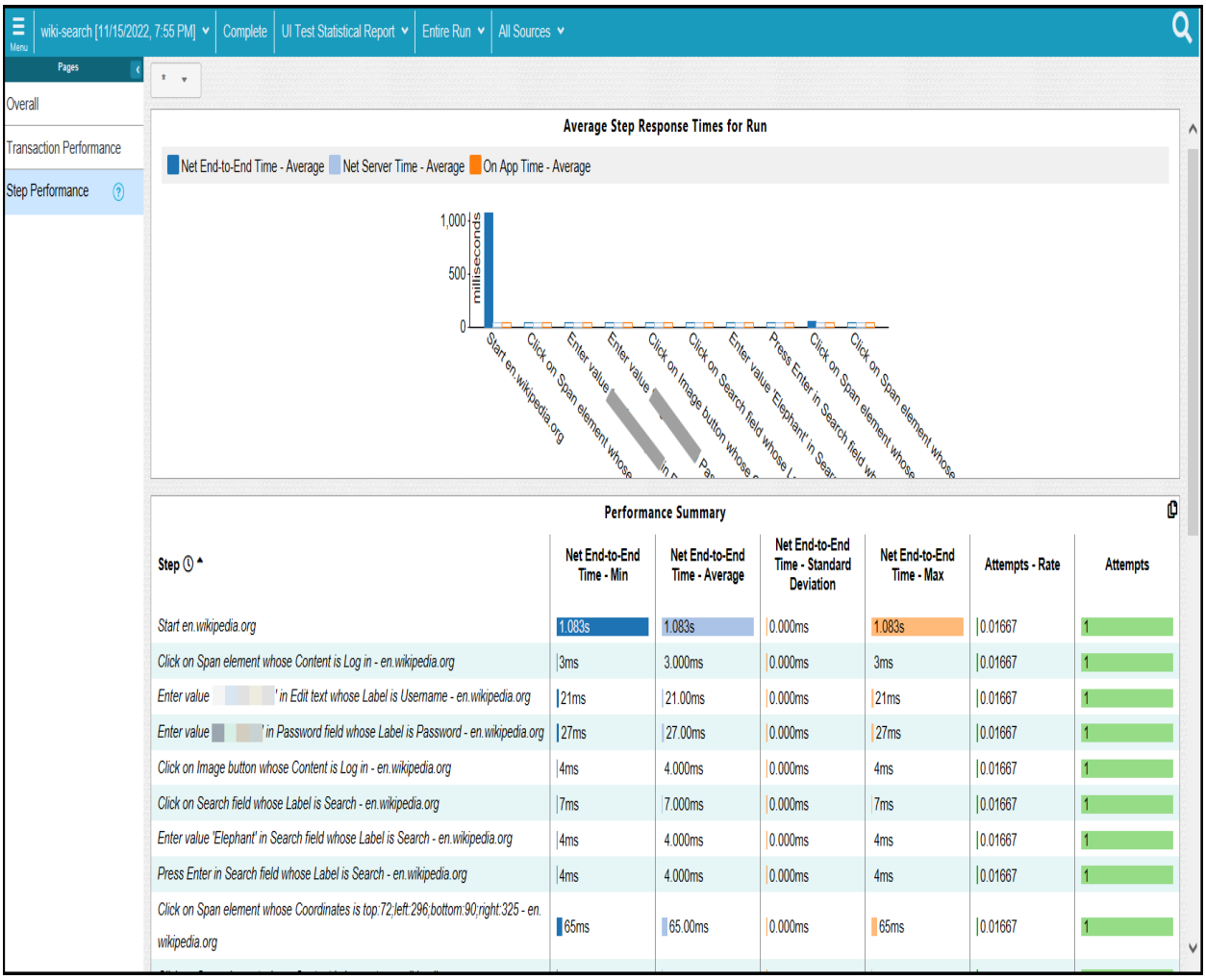


UI Test Statistical reports

UI Test Statistical reports contain performance data of the application and the response time for each step. By using this report, you can identify the steps where the response time is large and debug the application to identify the cause of the large response time. See UI Test Statistical report on page 1469.

A few screen captures of the UI Test Statistical report that is generated for the test scenario of this tutorial are as follows:

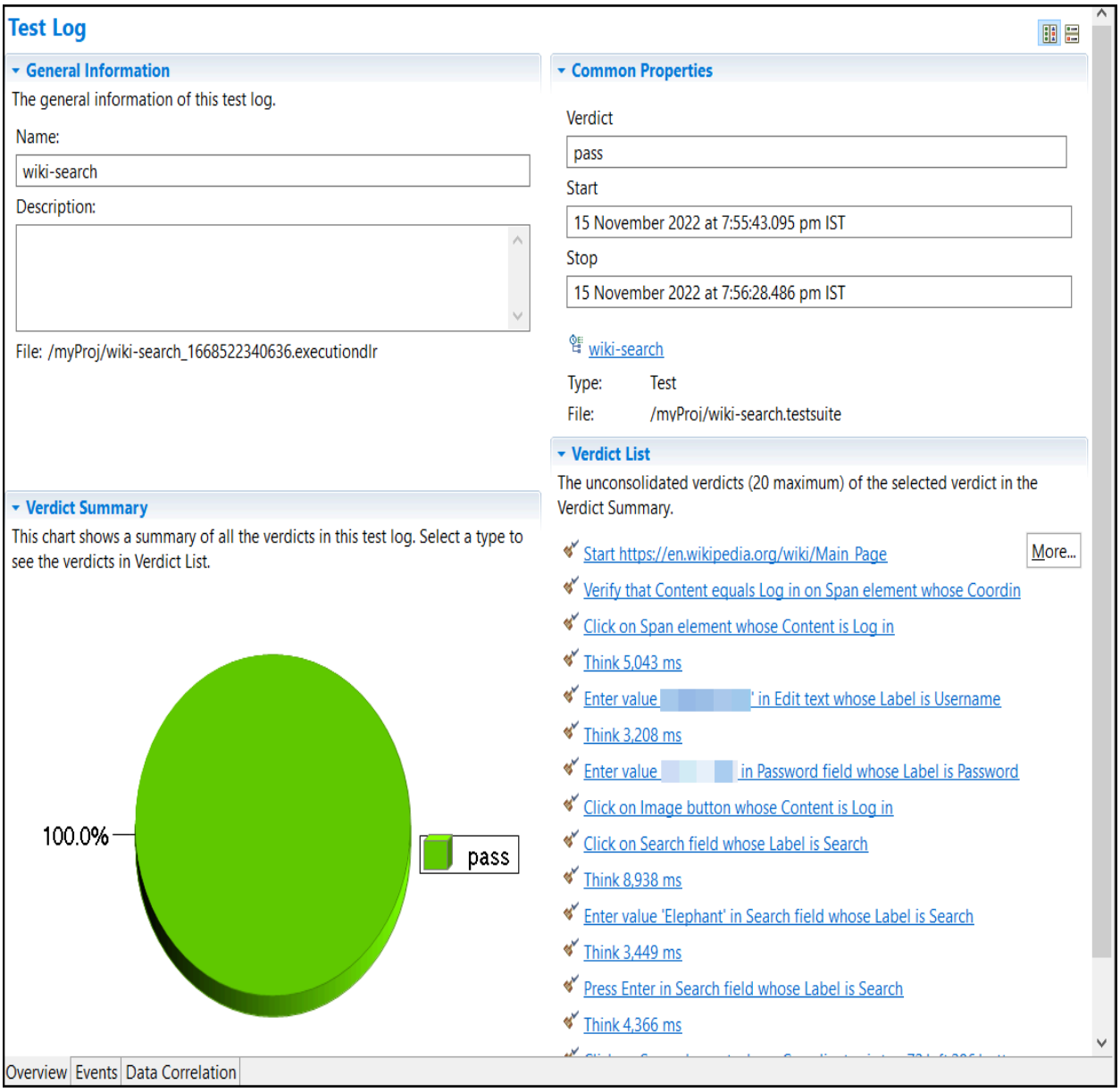

Menu
wiki-search [11/15/2022, 7:55 PM]
Complete
UI Test Statistical Report
Entire Run
All Sources
Pages
Overall
Transaction Performance
Step Performance
Run Summary
Counters
Executed Test
wiki-search
Elapsed Time
57.74s
Run Status
Complete
UI Test Health
Counters
UI Application Launches
1
UI Step Attempts
11
UI Verification Point Step Attempts
1
UI Action Step Attempts
9
UI Set Variable Step Attempts
0
UI In-Window Step Attempts
0
UI Custom JS Step Attempts
0
UI Other Step Attempts
1
Percent UI Verdicts passed
100.0%
Total UI Verdicts passed
11
Total UI Verdicts failed
0



Test Logs

Test Logs contain a record of events that occurred during a test run, and the status of each verification point. Rational® Functional TesterHCL OneTest™ UI generates the test log immediately after a test run. If a test run is unsuccessful, then you can use the information that is displayed in the test log to determine the reason for the test failure. See Test log overview on page 1493.

The following screen capture displays the Test Log that is generated for the test scenario of this tutorial:



Lesson 5: Splitting actions in a test

You can learn about splitting the actions of a test into test segments with different nodes. If a test contains many actions, you can split the test to create multiple nodes with meaningful test segments. You can then perform various actions such as inserting a loop, inserting a condition, and so on to the test segments of the selected nodes. You can also move a node from the existing test to a new test, and add the new test to a compound test that consists of various other tests that can be run sequentially.

Before you begin

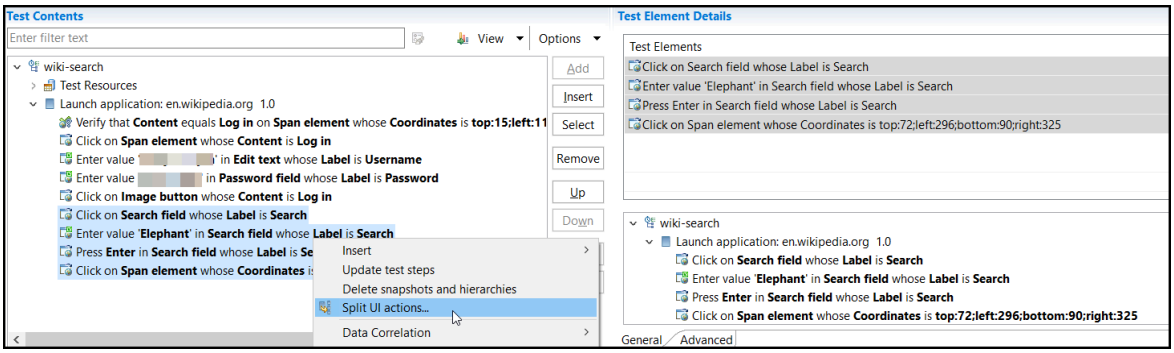
You must have completed the following tasks:

- Generated a Web UI test. See Lesson 1: Recording actions on an AUT on page 66.
- Read and been familiar with performing various actions on the nodes within a test. See Editing Web UI tests on page 573.
- Read and been familiar with the concept of compound tests. See Compound tests on page 737.
- Ensured that the **Test Contents** panel of the test editor window is displayed.

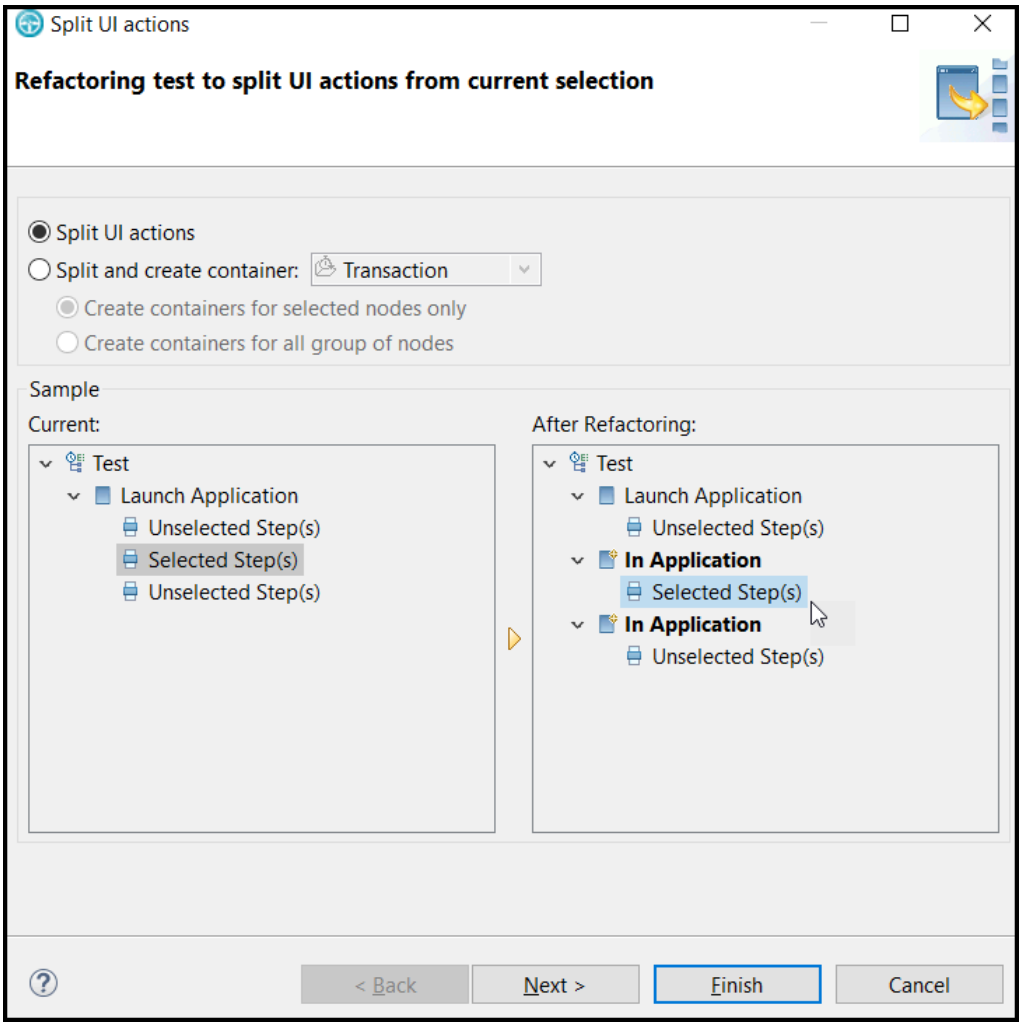
About this task

For the test scenario in this tutorial, you can split the actions that are related to logging in and searching into two different test segments. You can then group the actions related to searching and move them to a new test segment.

1. Select the actions from *search* to *logout* in the test that is displayed on the **Test Contents** panel.
2. Right-click the selected actions, and then click **Split UI actions**.

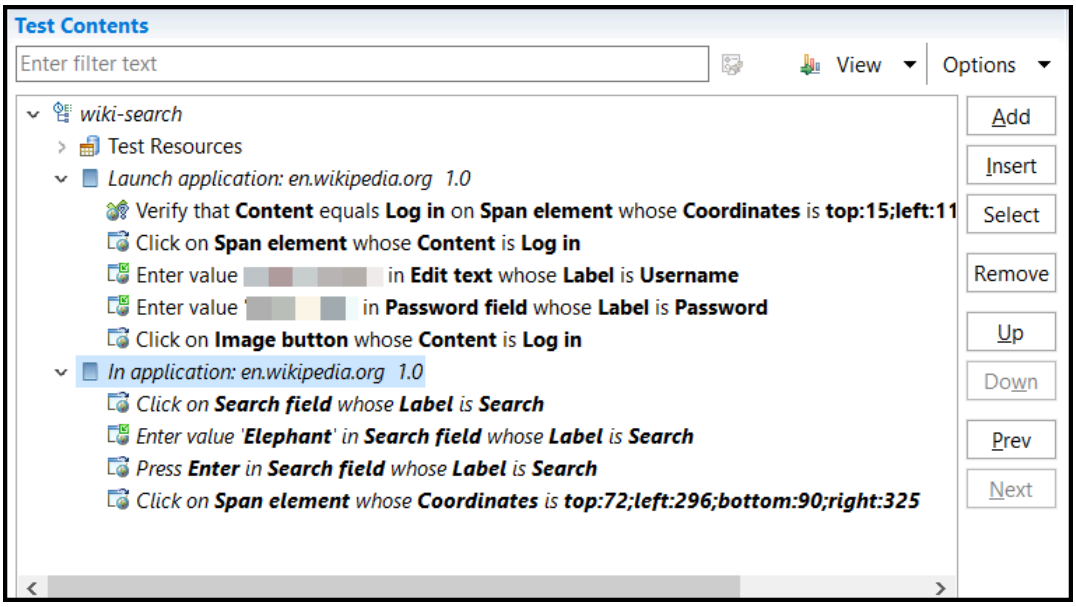


The **Refactoring test to split UI actions from current selection** dialog is displayed with the changes that you performed as a result of the split.



3. Click **Finish**.

   The *In application* node is displayed in the **Test Contents** panel, and the node contains all the actions that are split.



4. Click **File > New > New Test** to create an empty test.

   Result

   The **New Test** dialog is displayed.

5. Perform the following steps in the **New Test** dialog:

   a. Select *myProj* as the parent folder.

   b. Enter *wiki-search-animals* in the **File name** field, and then click **Next**.

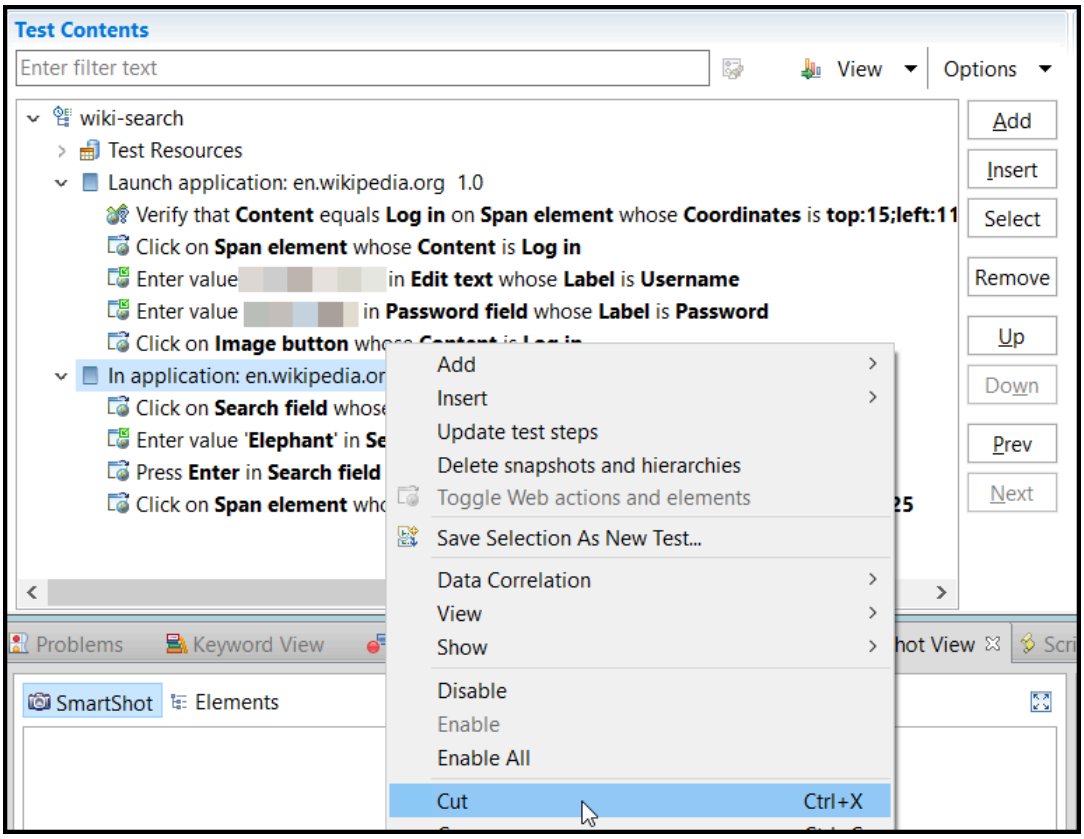
   c. **Optional:** Enter *Search for an animal's name* as the description, and then click **Next**.

   d. Select the **UI Test Feature** checkbox, and then click **Finish**.

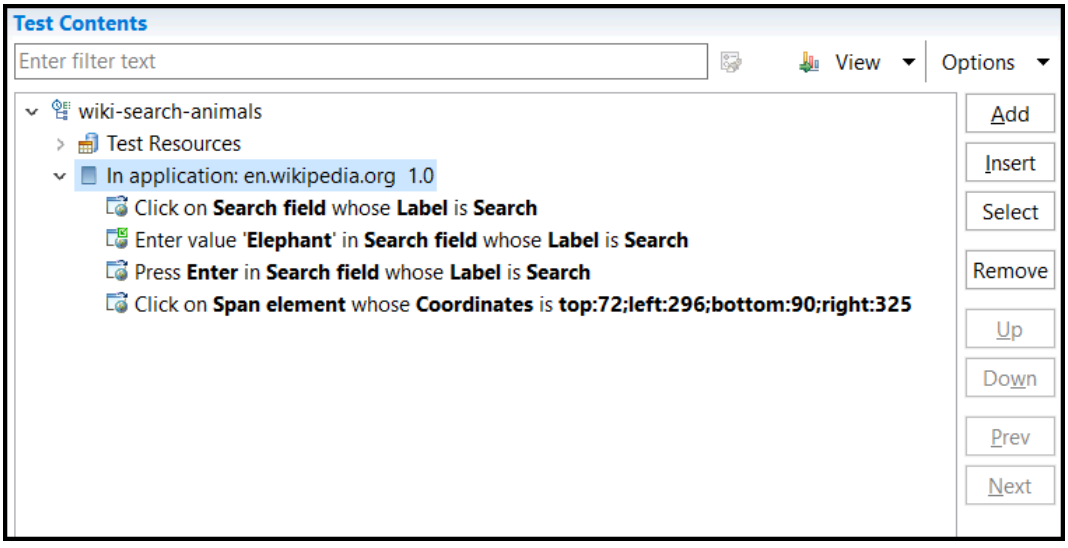
      Result

      A new test called *wiki-search-animals* is created and the **Test Contents** panel of the test is displayed.

6. Right-click the *In application* node in the **Test Contents** panel of the *wiki-search* test, and then click **Cut**.



7. Right-click the root node of the *wiki-search-animals* test, and then click **Paste**.
   The steps from the *In application* node of the *wiki-search* test are added to the *wiki-search-animals* test.



8. Click **File > Save**.

Results

You have learned to split the actions of a test into multiple test segments with different nodes and move a node from the existing test to a new test.

What to do next

You can learn about creating a dataset to associate it with a Web UI test. See Lesson 6: Creating a dataset on page 81.

Lesson 6: Creating a dataset

You can learn about creating a dataset for a test so that you can fetch the variable dataset values at the run time. A dataset is a collection of related data records that you can use to substitute the data in the test.

Before you begin

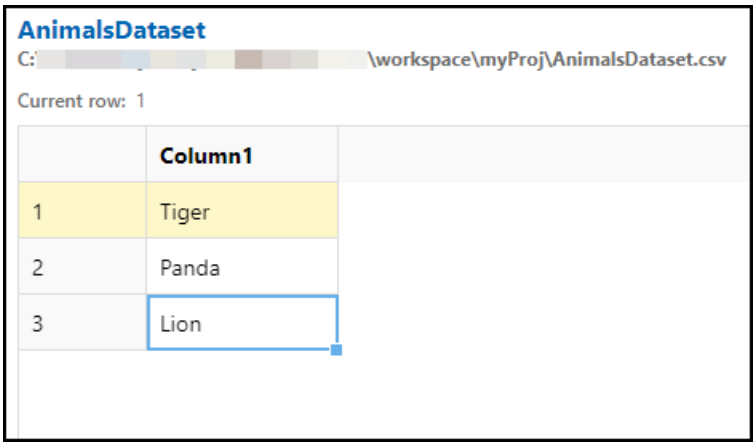
You must have completed the following tasks:

- Generated a Web UI test. See Lesson 1: Recording actions on an AUT on page 66.
- Read and been familiar with the concept of datasets. See Dataset overview on page 636.

About this task

For the test scenario in this tutorial, you can create a dataset that consists of three animal names, other than the one that you already entered in the application under test (AUT) while you record the test.

1. Right-click *myProj* in the **Test Navigator** panel, and then click **New > Dataset**.
2. Perform the following steps in the **New Dataset** dialog:
    a. Select *myProj* as the parent folder.
    b. Enter *AnimalsDataset* in the **File name** field, and then click **Next**.
    c. **Optional:** Enter *Dataset for animal names* in the **Description** field.
    d. Select *1* as **Columns** and *3* as **Rows** for the dimensions of the dataset, and then click **Finish**.
    e. Click **Yes** in the **Open dataset** dialog.
    Result
    The *AnimalsDataset* dataset is displayed in the CSV Editor in a browser.
3. Perform the following actions below **Column1** in the dataset:
    a. Enter *Tiger* in the first row, and then press the Down Arrow key.
    b. Enter *Panda*, and then press the Down Arrow key.
    c. Enter *Lion*.



4. Click the **Save** icon to save the dataset.

Results

You have learned to create a dataset for a test script.

What to do next

You can learn about associating a dataset with your test.

You can use the dataset to substitute the data in the test with the variable data from the dataset at the run time. See Lesson 7: Associating a dataset with a test on page 82.

Lesson 7: Associating a dataset with a test

You can learn about associating a dataset with a recorded test to add actual data to the test during the playback. The associated dataset is accessed during the playback of the test and the actual data in the test is substituted with the variable data in the dataset.

Before you begin

You must have completed the following tasks:

- Created a dataset. See Lesson 6: Creating a dataset on page 81.
- Ensured that the *wiki-search-animals* test is displayed in the **Test Contents** panel of the test editor window. See Lesson 5: Splitting actions in a test on page 76.

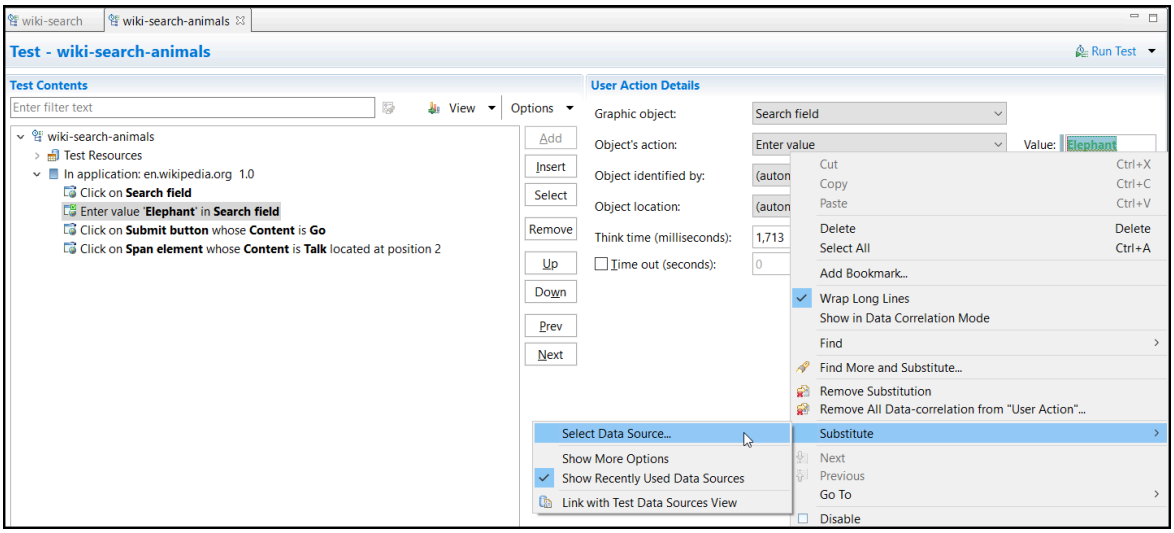
About this task

For the test scenario in this tutorial, you can associate the *AnimalsDataset* dataset with the *wiki-search-animals* test.

1. Select the *Enter value 'Elephant' in Search field* action in the *wiki-search-animals* test.
   Result

The details of the selected action are displayed in the **User Action Details** panel.

2. Right-click *Elephant* in the **Value** field, and then click **Substitute > Select Data Source**.



Result

The **Test Editor - Select Data Source** dialog is displayed.

3. Click **Yes** to display all the matching strings in the test that you can use as a data source to substitute the value of *Elephant* in the **Value** field.

Result

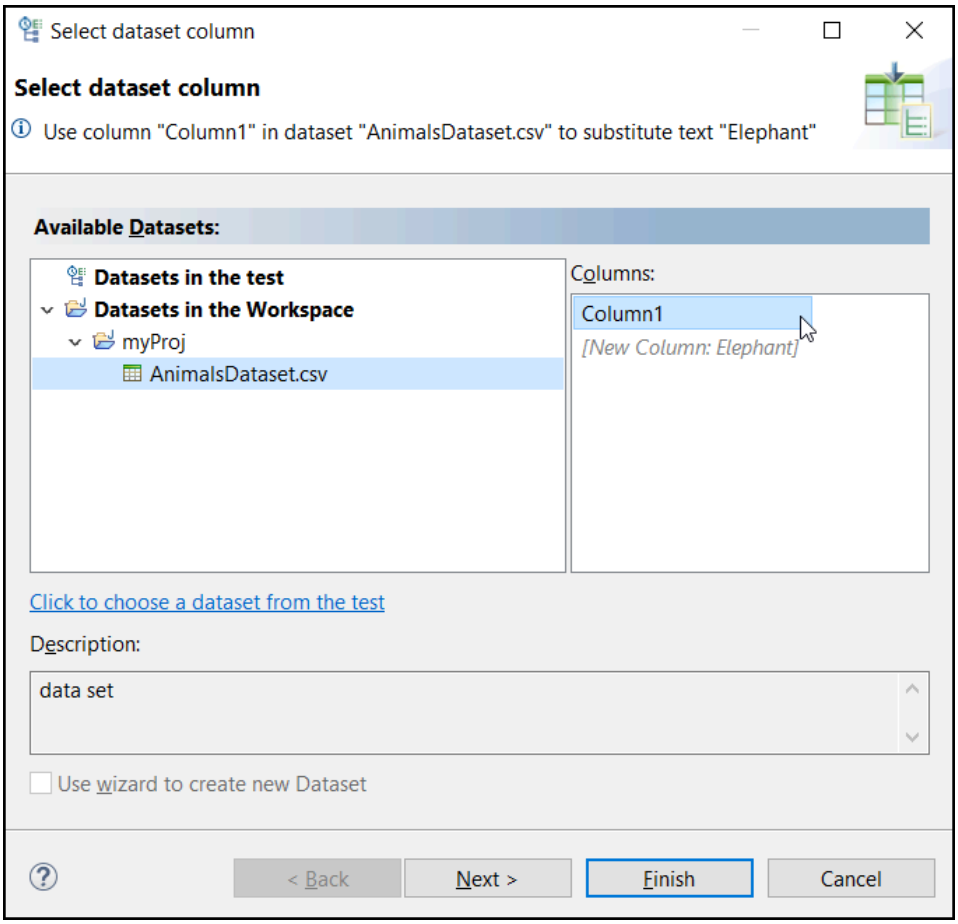
The **Select Data Source** dialog is displayed.

4. Click the **Add Dataset** icon  .

Result

The **Select dataset column** dialog is displayed.

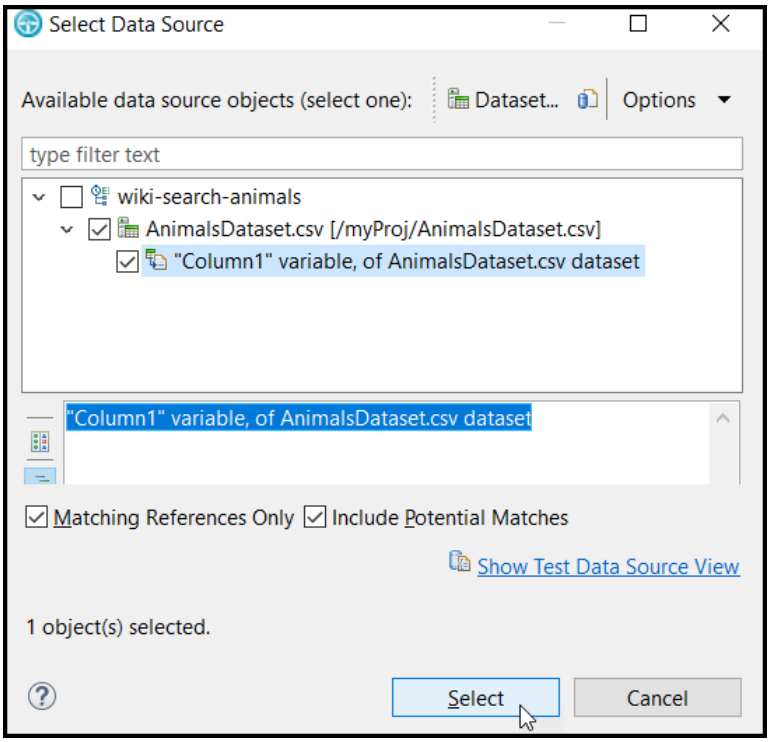
5. Select *AnimalsDataset.csv* in the **Available Datasets** panel, and then select *Column1* in the **Columns** panel.



6. Click **Finish**.

   Result

   The *AnimalsDataset* dataset is displayed in the **Select Data Source** dialog.
7. Ensure that *AnimalsDataset* dataset is selected, and then click **Select**.

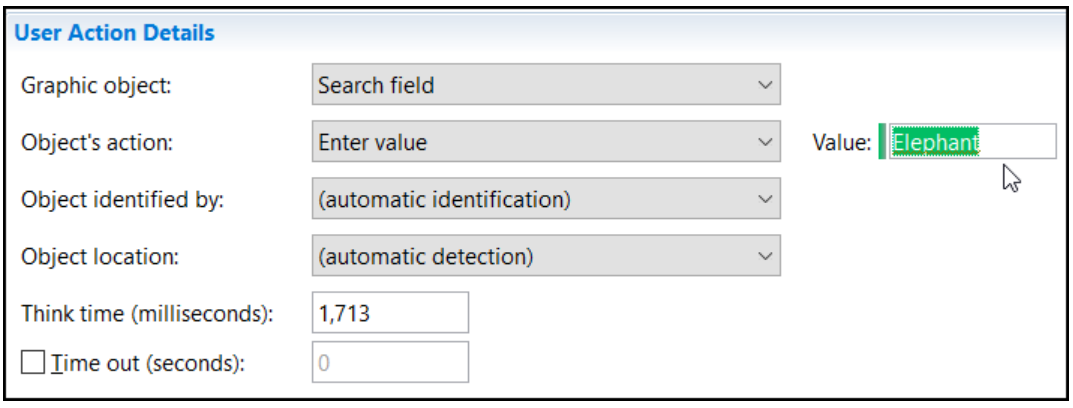


Result

The **Test Editor - Find and Substitute** dialog is displayed to confirm whether you want to view more substitutable locations in the test.

8. Click **Cancel**.

   The color of *Elephant* in the **Value** field of the **User Actions Details** panel changes to green. The change indicates that the selected dataset is successfully associated with the field.



9. Press Ctrl+S to save the changes.

Results

You have learned to associate a dataset with your test.

What to do next

You can learn about creating a compound test to combine various test scripts. See Lesson 8: Creating a compound test on page 86.

Lesson 8: Creating a compound test

You can learn about creating a compound test with multiple tests. Every individual test within the compound test performs a part of the main scenario. When you run the compound test, the individual tests are run in sequence.

Before you begin

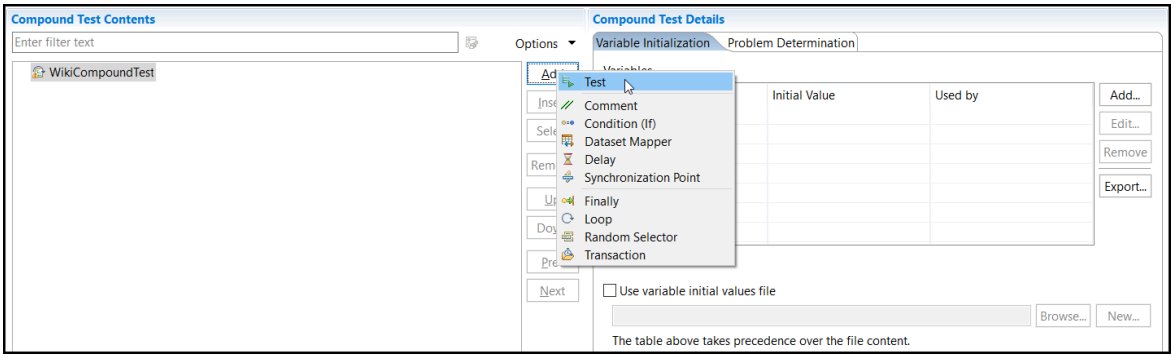
You must have completed the following tasks:

- Generated a Web UI test. See Lesson 1: Recording actions on an AUT on page 66.
- Read and been familiar with splitting actions in a test. See Lesson 5: Splitting actions in a test on page 76.
- Read and been familiar with the concept of compound tests. See Compound tests on page 737.

About this task

For the test scenario in this tutorial, you can create a compound test that consists of the *wiki-search* test followed by the *wiki-search-animals* test.

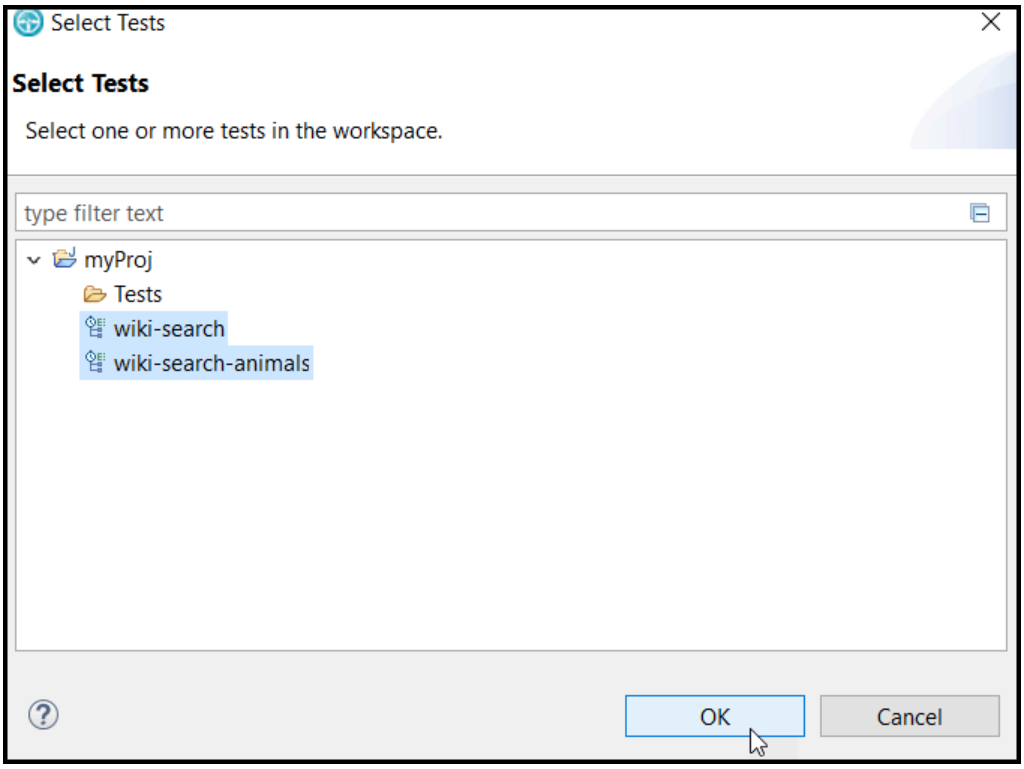
1. Click **File > New > Compound Test** in the **UI Test** perspective.
   Result
   The **Compound Test Name and Location** dialog is displayed.
2. Perform the following steps in the **Compound Test Name and Location** dialog:
   a. Select *myProj* as the parent folder.
   b. Enter *WikiCompoundTest* in the **File name** field, and then click **Finish**.
   Result
   The generated compound test is displayed in the test editor window.
3. Click **Add** in the **Compound Test Contents** panel, and then click **Test**.



Result

The **Select Tests** dialog is displayed.

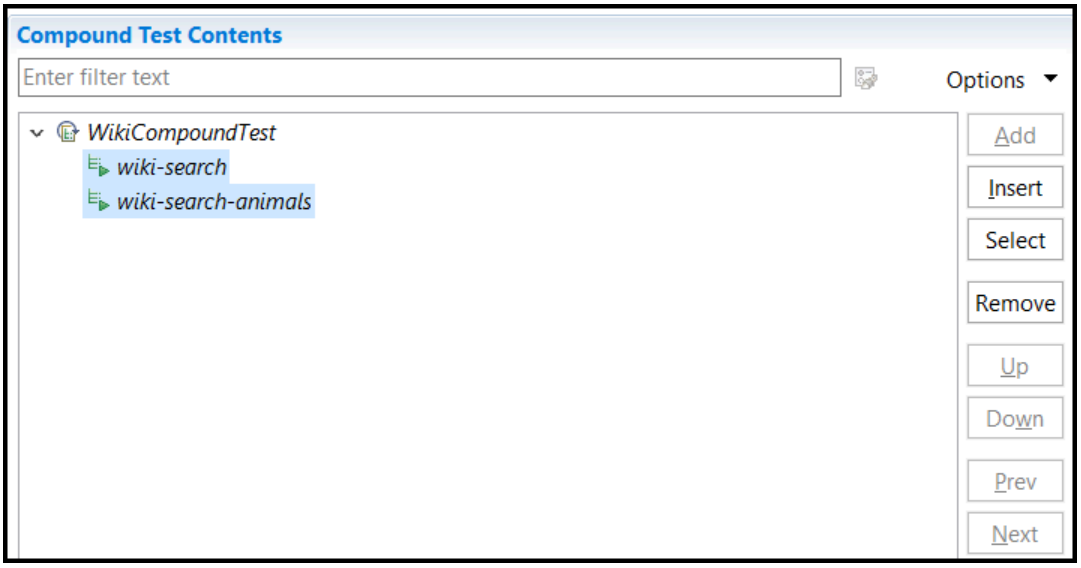
4. Select the *wiki-search* and *wiki-search-animals* tests from the *myProj* folder, and then click **OK**.



Result

The *wiki-search* and *wiki-search-animals* tests are displayed within the *WikiCompoundTest* test in the **Compound Test Contents** panel.

5. Click **File > Save** to save the changes.



Results

You have learned to create a compound test with multiple tests.

What to do next

You can learn about adding a loop to a compound test so that the tests within the compound test run sequentially and repeatedly for a specified number of time. See Lesson 9: Adding a loop on page 88.

Lesson 9: Adding a loop

You can learn about adding a loop to a test so that the test runs repeatedly for a specified number of time, while fetching values from an associated dataset. You can either select a count-based, a time-based, or an infinite loop for the test. You can set the loop to infinite if you want the test to fetch the data from all the rows in a dataset.

Before you begin

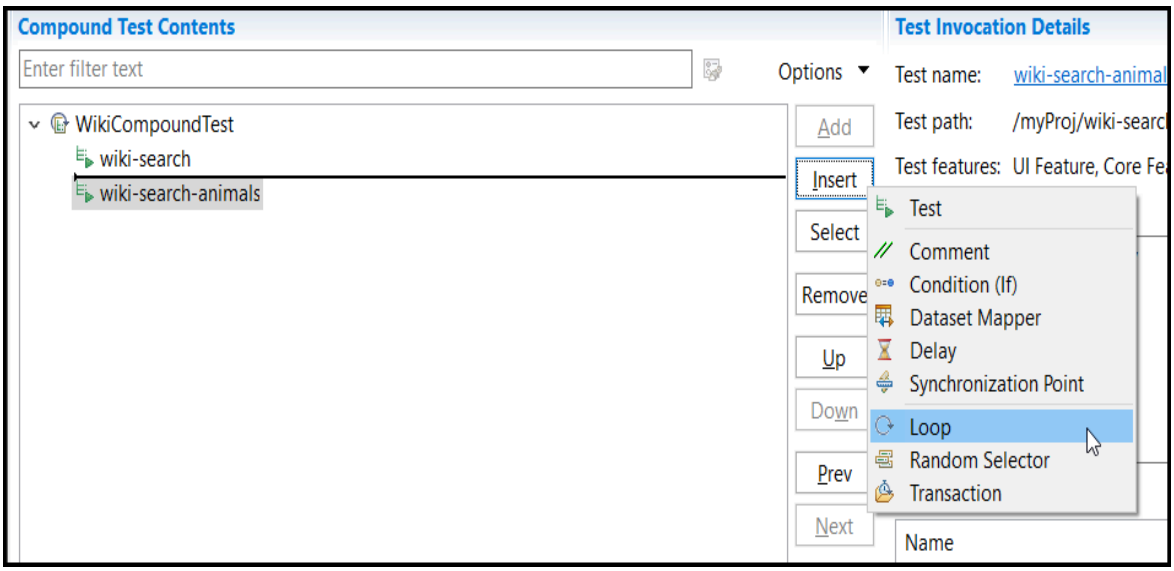
You must have completed the following tasks:

- Read and been familiar with running tests. See Lesson 3: Running a test on page 70.
- Read and been familiar with adding a loop to a test. See Adding a loop on page 587.
- Associated a dataset with the test. See Lesson 7: Associating a dataset with a test on page 82.
- Ensured that the *WikiCompoundTest* test is displayed in the **Compound Test Contents** panel of the test editor window. See Lesson 8: Creating a compound test on page 86.

About this task

For the test scenario in this tutorial, you can use *AnimalsDataset* as the associated dataset that has three entries. You can add a loop with three iterations to the test.

1. Select the *wiki-search-animals* test in the **Compound Test Contents** panel, and then click **Insert > Loop**.



The **New Loop** dialog is displayed.

2. Click **Yes** to move the *wiki-search-animals* test to a new loop.
   The *wiki-search-animals* test is displayed within a loop in the **Compound Test Contents** panel.
3. Enter *wiki-loop* as the loop name in the **Loop Details** panel.
4. Select the **Count-based** option, and then enter *3* in the **iterations** field.



5. Click **File > Save** to save the changes.
6. Click **Run Compound Test** to run the test.
   The **Confirm Perspective Switch** dialog is displayed.
7. Click **Switch** to run the test in the **Test Execution** perspective.
8. Select **Chrome** in the **Run Configuration** dialog.
9. Click **Finish**.
   The compound test run begins. The *wiki-search-animals* test runs three times while fetching the data from the *AnimalsDataset* dataset.

Results

You have learned to add a loop to a test so that the test runs repeatedly for a specified number of time.

What to do next

You can verify whether the recorded actions are successfully displayed in the test log and test reports after the test run is completed. You can verify the following:

- The associated dataset is successfully applied for the search action during the playback.
- The animal name in the test is substituted with the animal names that are listed in the dataset.
- The test is successfully repeated for the specified number of iterations.

Tutorials for testing in the Functional Test perspective

You can use the tutorials to get started with testing in the Functional Test perspective in Rational® Functional TesterHCL OneTest™ UI.

Prerequisites

Before you can get started with testing by using the tutorials, you must have completed the following tasks:

- Installed Rational® Functional TesterHCL OneTest™ UI. See Installation on page 201.
- Verified the system requirements specified for Rational® Functional TesterHCL OneTest™ UI. See System Requirements on page 13.

Be sure to start Rational® Functional TesterHCL OneTest™ UI on a new workspace.

Testing in the Functional Test perspective

You can find the following tutorials that you can use to get started with testing in the Functional Test perspective:

- Get started with functional testing using simplified scripts on page 90
- Create a functional test using Java scripts on page 104
- Perform a data-driven functional test using Java scripts on page 124
- Test Adobe Flex application on page 134
- Test GEF applications on page 138
- Extend Rational Functional TesterHCL OneTest UI capabilities using Proxy SDK on page 142

Get started with functional testing using simplified scripts

This tutorial introduces you to simplified test scripts and the application visuals and helps you get started with Rational® Functional TesterHCL OneTest™ UI for testing applications. It uses a sample Java™ application that is installed with the product for functional testing.

Learning objectives

During this tutorial, you learn to perform these tasks:

- Create functional test projects and record simplified test scripts
- Data-drive a test
- Create verification points
- Modify test scripts
- Work with application visuals

- Insert a Java custom code snippet
- Play back test scripts

Time required

60 minutes

Introduction: Get started with functional testing using simplified scripts

In this tutorial, you learn to create simplified test scripts, work with application visuals, and get started with functional testing. You review use cases for testing and performing basic functional testing operations. This tutorial uses the sample application provided with Rational® Functional TesterHCL OneTest™ UI in performing all the tasks.

The tutorial is divided into eight lessons that must be completed in sequence for the tutorial to work correctly.

Learning objectives

You learn how to perform these tasks:

- Create functional test projects
- Record simplified test scripts
- Data-drive a functional test
- Work with verification points
- Use application visuals
- Modify simplified test scripts
- Insert custom code
- Play back test scripts

Time required

This tutorial requires approximately 60 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Skill level

This is an introductory tutorial. Typically, users with little or no experience with Rational® Functional TesterHCL OneTest™ UI can perform the tasks.

Lesson 1: Set up Rational® Functional TesterHCL OneTest™ UI

IBM provides a Java™ Runtime Environment (JRE) that is installed and enabled for testing Java™ applications. Use this JRE for the tutorial. When you want to test your own Java™ or HTML applications, run the enabler and configure your environments and applications. For more information about these setup tasks, see the Getting Started with Rational® Functional TesterHCL OneTest™ UI wizard in the First Steps section of the product Welcome. For now, you do not need to do anything to use the preconfigured JRE to continue.

Set logging options

About this task

Rational® Functional TesterHCL OneTest™ UI provides several logging options. For this tutorial, use the HTML log.

1. Click **Window > Open Perspective > Other** to open the functional test perspective.
2. In the **Open Perspective** dialog box, select the **Functional Test** option.
3. To verify that HTML logging is set, click **Window > Preferences**.
4. In the left pane of the **Preferences** window, expand **Functional Test > Playback**, and click **Logging**.
5. Select **html** as the Log type, and then click **OK**.

   Show Me

Results

This setting opens the HTML log automatically after you play back a script.

Create a functional test project

About this task

Before you can start recording the test scripts, create a functional test project.

1. Click **File > New > Functional Test Project**.
2. In **Project name**, type `SimplifyTutorial` (no spaces).
3. In **Project location**, type `C:\FTproject`.
   The directory is created.
4. Click **Finish**. Show Me

Results

You can see the SimplifyTutorial project in the Functional Test Projects view, which is the left pane in the Functional Test perspective.

Enable simplified scripting and application visuals feature

About this task

You can generate simplified test scripts and Java™ test scripts. With the simplified test scripts and application visuals feature enabled, you can switch to Java™ scripting, if required but not vice versa. Before you start recording the scripts, enable the simplified scripting and the application visual features.

1. To verify that the feature is enabled, click **Window > Preferences**.
2. In the left pane of the **Preferences** window, expand **Functional Test**, and then click **Simplified scripting**.
3. On the **Simplified Scripting** page, select **Enable Simplified Scripting**.
4. On the **Application Visuals** page, select the **Enable capturing of application visuals**, **Insert Data Driven Commands**, and **Enable capturing of verification on test data** options.
5. Click **Apply**, and then click **OK**.

Lesson 2: Record a simplified test script

In this lesson, you record a simplified test script by using the recorder to test the Java application that is installed with the product.

Begin recording

1. To start recording, click the **Record a Functional Test Script** icon ( ) in the Functional Test toolbar.
2. Select the **SimplifyTutorial** project that you created in Lesson 1.
3. In **Script name**, type `Order`.
   Do not select the **Add script to Source Control**, if it is available.
4. Click **Next**.

   When you create a test script, a test dataset and other test assets are created. Use the defaults for Private Test dataset and Sequential. A private test dataset is associated with only one script and is not available to any other scripts. When you use the sequential order, the test script accesses dataset records in the order that they are arranged in the dataset.
5. Click **Finish**.
   Result

   The Rational® Functional TesterHCL OneTest™ UI window automatically minimizes, and the **Recording Monitor** is displayed.

   Learn more about Recording Monitor: The Rational® Functional TesterHCL OneTest™ UI Recording Monitor toolbar is displayed every time you begin recording a simplified script. You can minimize the monitor if you do not want to see the monitor on the screen. You can also resize the monitor. Click the **Display Monitor** icon ( ) to view the monitor messages. Leave the monitor displayed during this tutorial. The monitor displays messages for every action that you perform during your recording session, such as starting and pausing the recording, starting an application or browser, clicking within an application, inserting verification points, and inserting other items into the script.

Start the application and record the actions

About this task

Next, open the ClassicsJavaA application and record placing an order in the application.

1. To start the test application, click the **Start Application** icon ( ).
2. In the **Start Application** window, select ClassicsJavaA, and then click **OK**.
   Result

   The tutorial sample application, ClassicsCD, opens. If the recording monitor is in front of the application, you can drag it to the lower right corner of the screen.

3. Click the plus sign (+) next to Bach to open the list of CDs for sale by that composer, and then click **Violin Concertos**.
4. Click **Details** page to view the description of the album.
5. Click **Place Order**.
6. In the **Member Logon** window, keep the default settings of **Existing Customer** and Trent Culpito
7. Type `xxxx` in the **Password** field.
8. Click **OK**.
9. In the **Place an Order** window, type `1234 1234 1234 1234` in the **Card Number** field, and then type `12/12` in the **Expiration Date** field. Show Me

Lesson checkpoint

About this task

In this lesson you learned these tasks:

- How to start recording a test script
- How to use recording monitor
- How to open the test application

Lesson 3: Perform a data-driven test

In this lesson, you insert data-driven actions into the test script and populate a dataset with the data from the sample application.

About this task

A dataset is a collection of related data records. A dataset supplies data values to the variables in a test script during test-script playback.

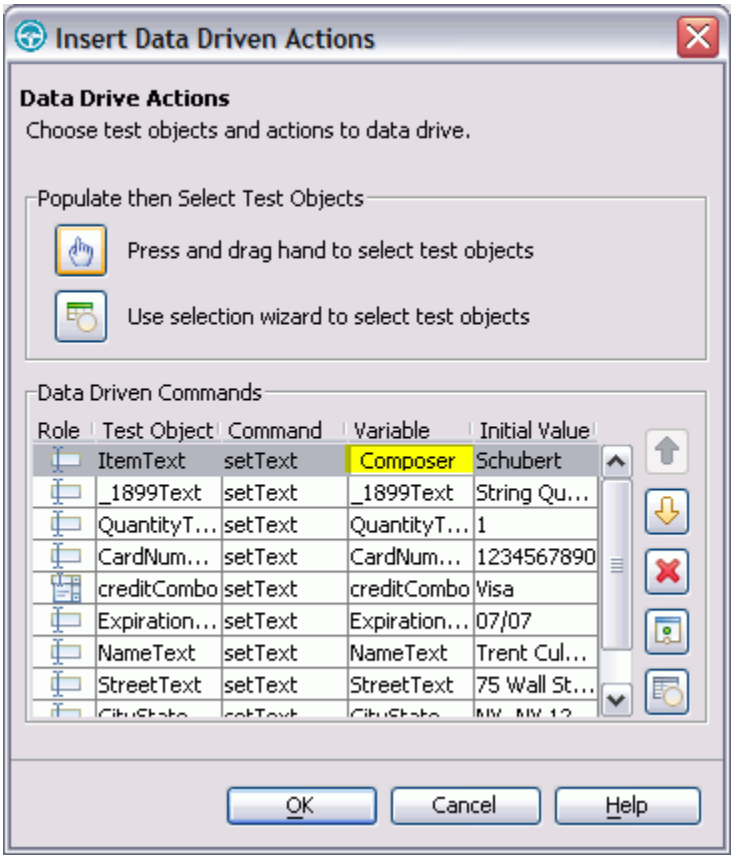
1. In the recording monitor toolbar, click **Insert Data Driven Commands** ( ).
   Result
   The recording pauses.
2. On the **Insert Data Driven Actions** page, drag the **Object Finder** ( ) to the title bar of the **Place an Order** window on the **ClassicsCD** application.
   Result
   The entire **Place an Order** window is outlined with a red border.
3. Release the mouse button.
   Result
   On the **Data Drive Actions** page, the **Data Driven Commands** table displays the information about the selected controls.

Add descriptive headings to the data

About this task

Now add descriptive headings to the dataset that you created. Descriptive headings make it easier to add data to the dataset.

1. In the **Data Driven Commands** table, change the **Variable** column by replacing the **Item** text with `Composer`.



2. Repeat sequentially, to replace each cell in the **Variable** column with a descriptive name for each heading in the **Variable** field. Use the text in the following variables list for descriptive names.

Note: Do not use spaces in **Variable** names. Typically, you review the application to determine the appropriate headings for each row, but we have done that for you in the following variables list. Use these names to replace variables in the presented order:

- Composer
- Item
- Quantity
- CardNo
- CardType
- ExpiryDate
- Name
- Street

- CityStateZip
- Phone

3. Click **OK**.

   Result

   The dataset has descriptive headings that make it easier to add data later. In a later lesson, you add data to the dataset after you finish recording the test script.

Lesson checkpoint

About this task

You learned these tasks:

- How to record a data-driven test script
- How to use datasets
- How to change the variable names in the dataset

Lesson 4: Create a verification point with a dataset reference

In this lesson, you create a verification point with a dataset reference to check that the price for the CD is correct in the ClassicsCD application.

About this task

What is a verification point?: A verification point captures object information and literal values from the application under test and stores it as the baseline for comparison during playback. When you play back a script, a verification point captures the object information again to compare it to the baseline and see whether any changes have occurred, either intentionally or unintentionally. Comparing the actual object information in a script to the baseline is useful for identifying possible defects.

You use a dataset reference instead of a literal value for the value that you are testing in the verification point. Using datasets with verification points gives you more flexibility to test realistic data with your test scripts.

1. On the **Recording** toolbar, click **Insert Verification Point or Action Command** ( ).

   Result

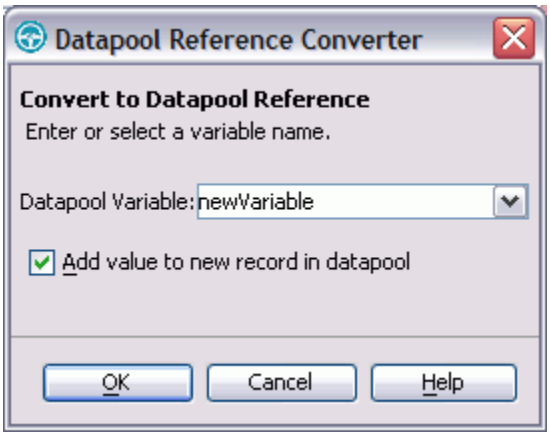
2. In the **Verification Point and Action** wizard, drag the **Object Finder** ( ) to `$14.99`, which is next to `Sub-Total` in the ClassicsCD application.

   Result

   The amount, `$14.99`, is outlined with a red border.
3. If the **Select an Action** page is not displayed, click **Next**.
4. On the **Select an Action** page, click **Perform Data Verification Point** to test whether the price of the CD changes.
5. Click **Next**.

6. On the **Insert Verification Point Data Command** page, type `Price` in **Verification Points Name**, and click **Next**.
7. On the **Verification Point Data** page toolbar, click **Convert Value to dataset Reference** ( ) to use a dataset instead of a literal value in a verification point. (If you cannot see **Convert Value to dataset Reference** on the toolbar, make the page larger by dragging a corner of the page).
   Result

   The **dataset Reference Converter** dialog box opens.



8. In **dataset Variable**, type `Price` to replace the **newVariable** value as the heading in the dataset.
9. Select **Add value to new record in dataset** to add the **Price** variable to the existing dataset record that you created in the previous exercise.
10. Click **OK**.
11. Click **Finish**.

Place the order and close the ClassicsCD application

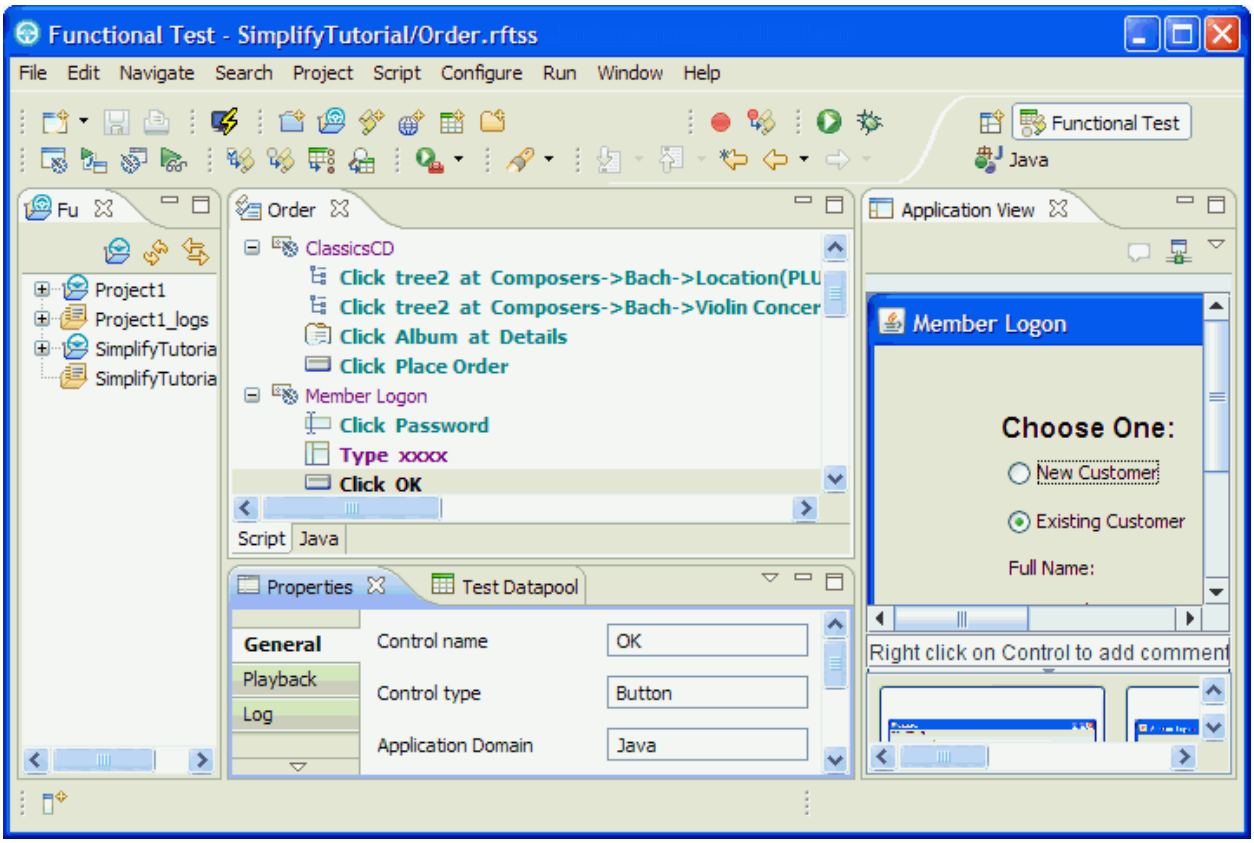
1. In the **ClassicsCD** application, click **Place Order** to place the order, and then click **OK** to close the message that confirms your order.
2. Click **X** in the upper right corner of the **ClassicsCD** application to close the application.

Stop recording

On the **Recording** toolbar, click **Stop Recording** ( ) to write all recorded information to the test script.

Results

The test script is displayed in the script editor window.



Lesson checkpoint

About this task

In this lesson, you learned about verification points, creating a verification point while recording, and converting a value to a dataset reference.

Lesson 5: Add data to the dataset

In this lesson, you add data to the dataset to test the ClassicsCD sample application by placing more orders for the CD.

About this task

To display the **Test dataset** editor, click **Window > Show view > Other**. In the **Show view** dialog box, expand **Functional Test**, click **Test dataset** and then click **OK**.

The dataset editor opens to the right of the script editor, and looks like this table:

	Composer	Item	Quantity	Card#	CardType	ExpDate	Name	Street	CityStZip	Phone	Price
0	Bach	Violin Concer-tos	1	1234 1234 1234 1234	Visa	12/12	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$14.99

1. Position your mouse pointer in the dataset editor, and then press **Enter** to add a row after the first row.

2. To save time, copy the data from row 0 in the dataset into the empty row that you created.
    a. Position the mouse pointer in the row 0 cell, right-click, and then click **Copy**.
    b. Position the mouse pointer in the row 1 cell, right-click, and then click **Paste**.
    c. Click **Yes** to paste the data into the empty row.
3. Change the value in the **Quantity**, **Card#**, **CardType**, and **ExpDate** columns to test the ClassicsCD sample application by placing more orders for the CD.
    a. In row 1, in the Quantity column, select the cell, and type `2`.
    b. In row 1, in the Card# column, select the cell, and type `9999 9999 9999 9999`.
    c. In row 1, in the CardType column, select the cell, and select `Amex` from the list.
    d. In row 1, in the ExpDate column, select the cell, and type `12/13`.

    Result

    The data in the dataset looks like this table:

	Composer	Item	Quantity	Card#	CardType	ExpDate	Name	Street	CityStZip	Phone	Price
0	Bach	Violin Concertos	1	1234 1234 1234 1234	Visa	12/12	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$14.99
1	Bach	Violin Concertos	*2*	*9999 9999 9999 9999*	*Amex*	*12/13*	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$14.99

4. In the **Test dataset** editor, click **X** to close the dataset editor, and then click **Yes** to save the changes you made to the dataset.

Lesson 6: Play back the test script

In this lesson, you play back the script and look at some parts of the application interface. Because the script you recorded is the active script, that script plays back when you click the playback button.

About this task

Each time you play back a script with an associated dataset, the script accesses one record in the dataset. When you create a dataset reference for a verification point, the verification point uses the dataset reference to access a variable in that record. During playback, the variable in the dataset is substituted for the dataset reference. The variable in the dataset is compared to the test results.

1. To play back the script, click **Run Functional Test Script** () on the toolbar.
2. In the **Select Log** window, keep the default log name **Order**, and then click **Next**.
3. Select the **Iterate Until Done** option from the **dataset Iteration Count** list to access all the records in the dataset; then click **Finish**.

    Result

    Rational® Functional TesterHCL OneTest™ UI window is minimized, and the playback monitor is displayed. As the script is executed, messages are displayed in the playback monitor. Rational® Functional TesterHCL OneTest™ UI plays back all of your recorded actions, such as the application starting, the actions you performed on the application, and enters data from the dataset.

When playback is finished, the HTML log displays the results of the test run. Each event listed in the log includes `Pass` in the event headings in green.

A log is a file that contains the record of events that occur while a script is played back. A log includes the results of all verification points that are run that can be used to test the application.

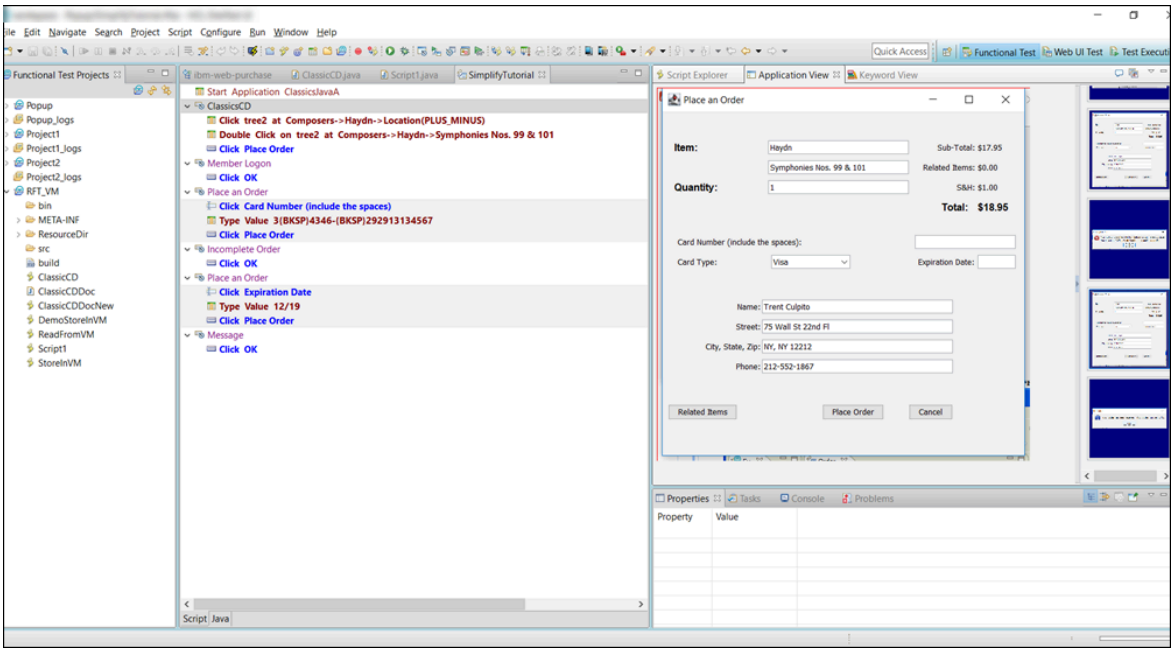
4. Close the log.

View the simplified script and the application visuals

About this task

After recording a script and playing it back, look at the Functional Test perspective in more detail.

1. If the Functional Test window is minimized, restore it.
   The generated simplified script is displayed in the Script editor.



   To the left of the Script editor is the Functional Test Projects view, which lists any functional test projects to which you are currently connected. All the scripts within each project are listed below the project name. This Projects view provides another way to navigate to different scripts. When you double-click a script in the Projects view, it opens in the script window and becomes the active script.

2. In the script editor, click the test line `Click PlaceOrder`.
   The application visual of the PlaceOrder is displayed in the Application View. The controls of the application and their properties are captured when you record the simplified script. The application visuals are displayed in the Application View. The application visual highlights the PlaceOrder control in blue. You can click each test line in the script and view the corresponding application visual in the Application View. The Thumbnails pane in the Application View displays the application visuals of all the test scripts in the project that are

captured while recording scripts. Notice that when you point the mouse over any of the thumbnails, the snapshot is zoomed and displayed.

3. Move the mouse pointer over any control in the displayed application visual, right-click, and select **Insert Comment** to insert any comments on the control.
   This feature is useful when you want to check the state of the control or make a note about the control for later reference.
4. Click **Java** editor to view the corresponding Java code of the test script.
   Notice that each test line of the simplified script is added as a comment to the corresponding Java code. This comment method makes it easier to map the simplified scripts and the Java code. Do not directly edit the Java code in the Java editor, because changes to the Java code are not shown in the simplified script. If you want to switch to Java scripting to use some of the functions not provided by simplified scripting, then you can use the Insert Custom Code feature and insert the required Java script.
5. Click **Script** to continue working with the simplified script in the script editor.

Lesson checkpoint

About this task

In this lesson you learned about the Functional Test perspective and how to play back a test script.

Lesson 7: Edit the simplified script by using the application visuals

In this lesson, you learn to edit the simplified script by using the application visuals.

About this task

The application controls and their data and property details are captured during recording. The captured details are displayed as application visuals in the Application View. You can modify the test script to test additional application controls or create or edit verification points by selecting the application controls in the application visuals without opening the application under test.

Insert a verification point by using the application visual

About this task

You create a verification point on the Composers tree to test whether all the composers and the CDs are listed. This verification point was not inserted when you recorded the test script. Instead of rerecording the script or opening the test application, you add the verification point to the test script from the application visual.

1. Select the test line `Click tree2 at Composers->Bach->Location((Plus_Minus)`(the second line in the test script).
   The application visual that highlights the Composers tree region in blue is displayed in the Application View.
2. Point to the Composers tree region. The region is highlighted in red.
3. Right-click, and select **Insert Verification Point > Data Verification Point**.
   The test line `Verify data in tree2` is added to the script editor (after the second test line).

Add an additional control to the test

About this task

In the **Member Logon** window, you did not record the **Remember Password** field for testing. Add the missing **Remember Password** control to the test script for testing.

1. Select the test line `Type xxxx` in the script editor.
   The **Member Logon** application visual is displayed in the Application View.
2. Point to the **Remember Password** control in the application visual.
   The **Remember Password** control is highlighted in red.
3. Right-click and select **"Insert Remember Password" control > select**.
   The `Select Remember Password` test line is inserted to the test script.

Modify the test line in the script editor and the properties

About this task

You can modify the test line in the script editor and also specify details such as the playback parameters and log information for the test line execution in the Properties view. Now, disable the test line for viewing the details of the album and specify the information that must be displayed in the log during the execution of the data verification for the list of composers control.

1. Select the test line `Click Album at Details` in the script editor.
2. Right-click and select **Enable/Disable Action** to disable the test line.
   This test line is not executed during the next playback.
3. Select the test line `Verify data in tree2` that was inserted to the script using the application visual.
4. Click **Log** page in the **Properties** view and select **Control Snapshot** to view the state of the control during execution. This snapshot is displayed in the playback log.

Lesson 8: Insert Java custom code

You can switch to Java scripting if you want to insert Java codes to perform additional operations such as extending an API or any functions that cannot be performed directly in the simplified script editor.

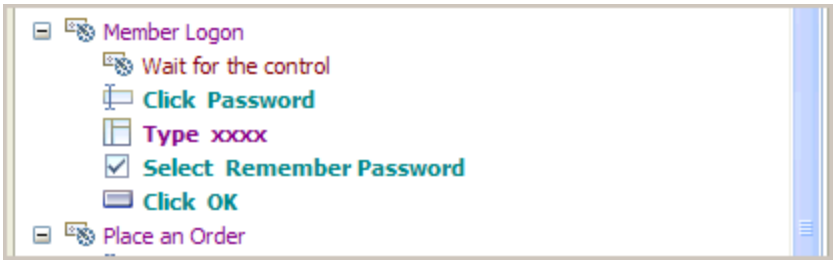
About this task

To use both the simplified script and the Java scripting, you must use the Insert Java Code Snippet or Insert Java Method feature available in the simplified script editor and switch to Java scripting. If you modify the Java script directly without using these features, the Java script changes are lost and the simplified script is executed during playback.

In this lesson, you insert a Java code snippet so that Rational® Functional TesterHCL OneTest™ UI waits until the **Password** control in the **Member Logon** window is displayed in the application during playback and then test the control.

You can enable the option to wait for the control to be displayed for a test line in the **Playback** page in the **Properties** view. But in this tutorial, to understand the process of inserting a custom Java code, you perform the following steps:

1. Select the test line `Click Password` in the script editor.
   (The first test line in the Member Logon group.)
2. Right-click and select **Insert Java Code Snippet**.
   Result
   The test line `Click here to tag the Java snippet test line` is inserted after the `Click Password`.
3. Select the inserted test line and replace the test line text by typing `Wait for control`.
4. Drag the `Wait for the control` test line and drop it above the `Click Password` test line so that Java code is executed before the password control is tested.
   Result



5. Click **File > Save** to save the simplified script.
6. Click **Java** editor that is displayed next to the **Script** editor.
   Notice that `Wait for control` is displayed as comments with the start and the end point for inserting the Java code in the Java editor.
7. Type the Java code `password().waitForExistence();` within the start and the end comment section.
   Result

```
setSimplifiedScriptLine(10);
 //Wait for the control
// BEGIN custom code Wait for the control
password().waitForExistence();
// END custom code Wait for the control
```

8. Click **File > Save** to save the Java script.

Play back the script

About this task

Play back the test script and verify the results of the modified test script.

1. To play back the script, click **Run Functional Test Script** ( ) on the toolbar.
2. In the **Select Log** window, keep the default log name **Order**, and then click **Next**.
3. Select the **Iterate Until Done** option from the **dataset Iteration Count** list to access all the records in the dataset; then click **Finish**.
   Result

Rational® Functional TesterHCL OneTest™ UI plays back the modified script. During playback, notice that the click action on the Details page of the Album is not executed.

When playback is finished, the HTML log displays the results of the test run. Each event listed in the log includes `Pass` in the event headings in green. You can also view the snapshot of the composers list.

Lesson checkpoint

About this task

In this lesson you learned how to modify the simplified test script and insert a Java custom code snippet to a simplified script.

Summary: Get started with functional testing using simplified scripts

This tutorial has shown you how to set up Rational® Functional TesterHCL OneTest™ UI for testing, recording and playing back simplified scripts, creating verification points, performing data-driven tests, and editing the scripts using the application visuals.

Lessons learned

By completing this tutorial, you learned how to:

- Create a functional test project and record simplified test scripts
- Create verification points
- Perform a data-driven test
- Use application visuals and edit the script
- Insert a Java custom code snippet
- Play back test scripts

Create a functional test using Java scripts

This Rational® Functional TesterHCL OneTest™ UI tutorial walks you through the major use cases for creating and playing back functional tests that use Java scripts. This comprehensive tutorial uses a sample Java™ application that is installed with the product.

Learning objectives

After completing this tutorial, you will be able to do the following tasks:

- Create a functional test project and record a Java test script
- Work with verification points, object maps, and regular expressions in Java test script
- Use a comparator to update a verification point in the Java test script
- Play back a Java test script
- Perform regression tests with Java test scripts

Time required

45 minutes

Related information

Perform a data-driven functional test using Java scripts on page 124

Sample project to test a Java application on page 180

Introduction: Create a functional test using Java scripting

This tutorial teaches you how to get started using Rational® Functional TesterHCL OneTest™ UI and walks you through the major use cases for testing and performing basic operations. This tutorial uses the sample application provided with Rational® Functional TesterHCL OneTest™ UI to perform all the tasks.

The Rational® Functional TesterHCL OneTest™ UI tutorial is divided into 10 lessons that must be completed in sequence for the tutorial to work properly.

Learning objectives

After completing this tutorial, you will be able to:

- Create a functional test project and record a Java test script
- Work with verification points, object maps, and regular expressions
- Use a comparator to update a verification point
- Play back a script
- Perform regression tests

Time required

This tutorial should take approximately 45 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Prerequisites

This is an introductory tutorial. You should be able to perform the tasks with little or no experience with Rational® Functional TesterHCL OneTest™ UI.

Lesson 1: Set up Rational® Functional TesterHCL OneTest™ UI

IBM® provides a Java™ Runtime Environment (JRE) that is installed and enabled for testing Java™ applications. Use this JRE for the tutorial. When you want to test your own Java™ or HTML applications, you must run the enabler and configure your environments and applications. For more information on these set-up tasks, see the Getting Started with Rational® Functional TesterHCL OneTest™ UI wizard in the First Steps section of the product Welcome. For now you do not need to do anything to use the preconfigured JRE to continue.

About this task

Start Rational® Functional TesterHCL OneTest™ UI and then perform the following tasks before you record your first test script.

Set logging options

About this task

Rational® Functional TesterHCL OneTest™ UI provides several logging options. We will use the HTML log.

1. Click **Windows > Open Perspective > Other** to open the functional test perspective. In the Open Perspective dialog box, select the **Functional Test** option.
2. To verify that HTML logging is set, click **Window > Preferences**.
3. In the left pane of the Preferences window, expand **Functional Test**, then **Playback**, and click **Logging**.
4. Select **html** as the **Log type**.
5. Click **OK**.

Results

This setting opens the HTML log automatically after you play back a script.

Disable simplified scripting and application visuals feature

About this task

Rational® Functional TesterHCL OneTest™ UI provides you the option to generate simplified test scripts and Java test scripts. If you are familiar with Java scripting, you can disable the simplified scripting and application visuals feature and start recording the test script. In this tutorial, we will work with Java test scripts. Before you start recording the scripts, disable the simplified scripting and application visual features.

1. To verify whether the feature is enabled, click **Window > Preferences**.
2. In the left pane of the **Preferences** window, expand **Functional Test**, then **Simplified scripting**.
3. In the **Simplified Scripting** page, clear the **Enable Simplified Scripting** checkbox.
4. In the **Application Visuals** page, clear all the options listed in the page for the application visuals.
5. Click **Apply** and then **OK**.

Create a functional test project

About this task

Before you can start recording, you must create a functional test project.

1. In the Rational® Functional TesterHCL OneTest™ UI menu, click **File > New > Functional Test Project**.
2. Under **Project name**, type `FTtutorial` (no spaces).
3. Under **Project location**, type `C:\FTproject`.
   Rational® Functional TesterHCL OneTest™ UI creates this directory.
4. Click **Finish**.

Results

The FTtutorial project is now visible in the Functional Test Projects view, which is the left pane in the Functional Test perspective.

Lesson 2: Record a script

In this lesson, you will record a script using the Rational® Functional TesterHCL OneTest™ UI Recording Monitor.

Begin recording

About this task

You are now ready to begin recording.

1. To start recording, click the **Record a Functional Test Script** button in the Functional Test toolbar.
2. Select the FTtutorial project that you just created.
3. In the **Script name** field, type `Classics` (the name of the application you will be using).
4. Do not select the **Add script to Source Control** option if it is available.
5. Click **Finish**.

   Result

   The Rational® Functional TesterHCL OneTest™ UI window automatically minimizes, and the Recording Monitor is displayed.

   Learn more about Recording Monitor: The Rational® Functional TesterHCL OneTest™ UI Recording Monitor is displayed every time you begin recording. You can minimize the monitor if you don't want it to be visible on the screen, and you can also resize it. You can also click the **Display Toolbar Only** button (), which hides the recording monitor and shows only the toolbar. Click the **Display Monitor** button () to bring it back. Leave the monitor displayed during this tutorial. The monitor displays messages for every action performed during your recording session, such as starting and pausing the recording, starting an application or browser, clicking within an application, inserting verification points, and inserting other items into the script.

6. Click the **Monitor Message Preferences** toolbar button . You can use these options any time to control the appearance of the text in the monitor.
7. Click **Cancel**.
8. Click the **Insert Script Support Commands** toolbar button .

   Result

   This opens the Script Support Functions window, which allows you to call another script, insert a log entry, insert a timer, insert a sleep command (a delay), or insert a comment into your script.

9. Click **Close**.

Start the application

1. To start the test application, click the **Start Application** toolbar button  .
2. In the Start Application window, select **ClassicsJavaA** and then click **OK**.

   Result

   The Rational® Functional TesterHCL OneTest™ UI Tutorial sample application, ClassicsCD, opens. If the Recording Monitor is in front of the application, you can drag it to the lower right corner of the screen.

Record actions

About this task

You are going to record placing an order in this application.

1. Click the **+** next to **Haydn** to expand the folder in the **Composers** tree.
2. In the list, click **Symphonies Nos. 94 & 98**.
3. Click the **Place Order** button.
4. In the Member Logon window, keep the default settings of **Existing Customer** and **Trent Culpito**. Do not click either of the password fields at this time.
5. Click **OK**.
6. In the **card number** field, enter a credit card number. You must use the valid format of four sets of four digits here, for instance, `7777 7777 7777 7777`.
7. In the**expiration date** field, enter a valid format expiration date, `06/09`.

   Note: Date can be same or later than the system date.

8. Click **Place Order**.
9. Click **OK** in the order confirmation message window.

Lesson 3: Create verification points

In this lesson, you will record verification points to test objects. Verification points verify that a certain action has taken place, or verify the state of an object.

About this task

You can create a Properties verification point, Image verification point or nine types of Data verification points. When you create a verification point, you capture information about an object in the application to establish baseline information for comparison during playback.

Create a data verification point

About this task

You will record a Data verification point to capture the tree of composers.

1. In the Recording Monitor, click the **Insert Verification Point or Action Command** button .
2. In the Select an Object page of the Verification Point and Action Wizard, clear the **After selecting an object advance to next page** option if it is selected.
3. Use the Object Finder) to select the Composers tree in the application. Click the **Object Finder** and drag it over the tree. While holding down the mouse button, you will see that the entire tree is outlined with a red border and the object name is displayed `javax.swing.JTree` in a screen tip next to the red border. When you release the mouse button to make the selection, notice that the recognition properties for the object are listed in the grid at the bottom of the Select an Object page.
4. Click **Next**.
5. In the Select an Action page, make sure **Perform Data Verification Point** is selected and click **Next**.
6. In the Insert Verification Point Data Command page, in the **Data Value** field, select the **Tree Hierarchy** test. This test captures information about the entire tree hierarchy.
7. In the Verification Point Name field, type `Classics_tree` and click **Next**.
8. The Verification Point Data page displays the captured data in a grid in the right pane. If a check mark appears in the box beside an item, that item will be tested. By default, all items are selected. Leave them checked. If they are not selected, click the **Check All** button.
9. Click **Finish**.

Create an image verification point

About this task

You can insert an image verification point to confirm that the appropriate album is displayed for the selected CD.

1. In the Recording Monitor, click the **Insert Verification Point or Action Command** button .
2. In the Select an Object page of the Verification Point and Action Wizard, clear the **After selecting an object advance to next page** option if it is selected.
3. Use the Object Finder to select the Album image in the application. Click the **Object Finder** and drag it over the album image. While holding down the mouse button, you will see that the album image outlined with a red border and the object name is displayed `javax.swing.JLabel` in a screen tip next to the red border. When you release the mouse button to make the selection, notice that the recognition properties for the object are listed in the grid at the bottom of the Select an Object page.
4. Click **Next**.
5. In the Select an Action page, select**Perform Image Verification Point** and click **Next**.
6. In the Insert Image Verification Point Command page, type `Album_image` as the **Verification Point Name**.
7. Make sure that the option **Select full image** is selected and click **Next**.
8. The Verification Point Data page displays the captured image in the right pane. Click **Finish**.

Create a properties verification point

About this task

You can now insert a different verification point to confirm that the order is for the correct customer. A Properties verification point captures the text in the confirmation screen.

1. In the ClassicsCD application, click **Order > View Existing Order Status**. Do not click either of the password fields at this time.
2. Click **OK**.
   You will test the label "Order for Trent Culpito" in the View Existing Orders window.
3. In the Recording Monitor, click the **Insert Verification Point or Action Command** button .
4. In the Select an Object page, select the **After selecting an object advance to next page** option.
5. Drag the **Object Finder** over the label "Order for Trent Culpito" to select it. While holding down the mouse button, note that the label is outlined with a red border and the object name is displayed `javax.swing.JLabel`). After you select the object, the Select an Action page opens because you selected the **advance to next page** option.
6. Select **Perform a Properties Verification Point**, which is the second action from the top, and then click **Next**.
7. On the Insert Properties Verification Point Command page, confirm that the **Include Children** field is set to **None**.
8. Under **Verification Point Name**, accept the suggested default.
9. Leave the **Use standard properties option** selected and then click **Next**.
   On the Verification Point Data page, the test object properties and their values are displayed in a grid format. You can choose which properties to test in the Property column and can edit the property values in the Value column.

   Learn more about selecting object properties: By default, none of the properties are selected. To test object properties, choose the properties that you want to test by selecting each property. The properties you select are tested each time you play back a script with this verification point. You can select all properties in the list by clicking the **Check All** toolbar button above the grid. Use the **Uncheck All** button to clear all properties. For best results when using a Properties verification point, test only the properties you are interested in. In this case, only the **text** property is of interest to determine whether the order is for the correct customer.

10. In the Property column select the **text**, **opaque**, and **visible** properties to test them during playback. You may have to click the check box twice for the selection to persist.
11. Click **Finish**.
12. In the ClassicsCD View Existing Orders window, click **Close**.

Test the password fields

About this task

Now let us place another quick order to test the password fields that we did not test earlier.

1. Expand the **Haydn** folder in the composers tree.
2. Click **Symphonies Nos. 94 & 98**.
3. Click the **Place Order** button.
4. In the Member Logon window, keep the default settings of **Existing Customer** and **Trent Culpito**.
5. This time, type `xxxx` in the **Password** field.

6. Select the **Remember Password** option.
7. Click **OK**.
8. Type a valid **card number** number and **expiration date**, for instance, `7777 7777 7777 7777`, expiration `06/09`.
9. Click **Place Order**.
10. Click **OK** in the order confirmation message box.
11. Close the ClassicsCD application by clicking the **x** button.
12. Click the **Stop Recording** button) on the Recording toolbar.

Results

When you stop recording, Rational® Functional TesterHCL OneTest™ UI closes the recording monitor and then writes your script and object map to your project directory. The Rational® Functional TesterHCL OneTest™ UI window is restored and the script is displayed in the main window.

Lesson 4: Play back the script

In this lesson, you will play back the script and look at some parts of the Rational® Functional TesterHCL OneTest™ UI interface. Because the script you just recorded is the active script, that script will play back when you click the playback button.

1. To play back the script, click the **Run Functional Test Script** button on the Functional Test toolbar.
2. In the Select Log window, keep the default log name **Classics** and then click **Finish**.
   Result

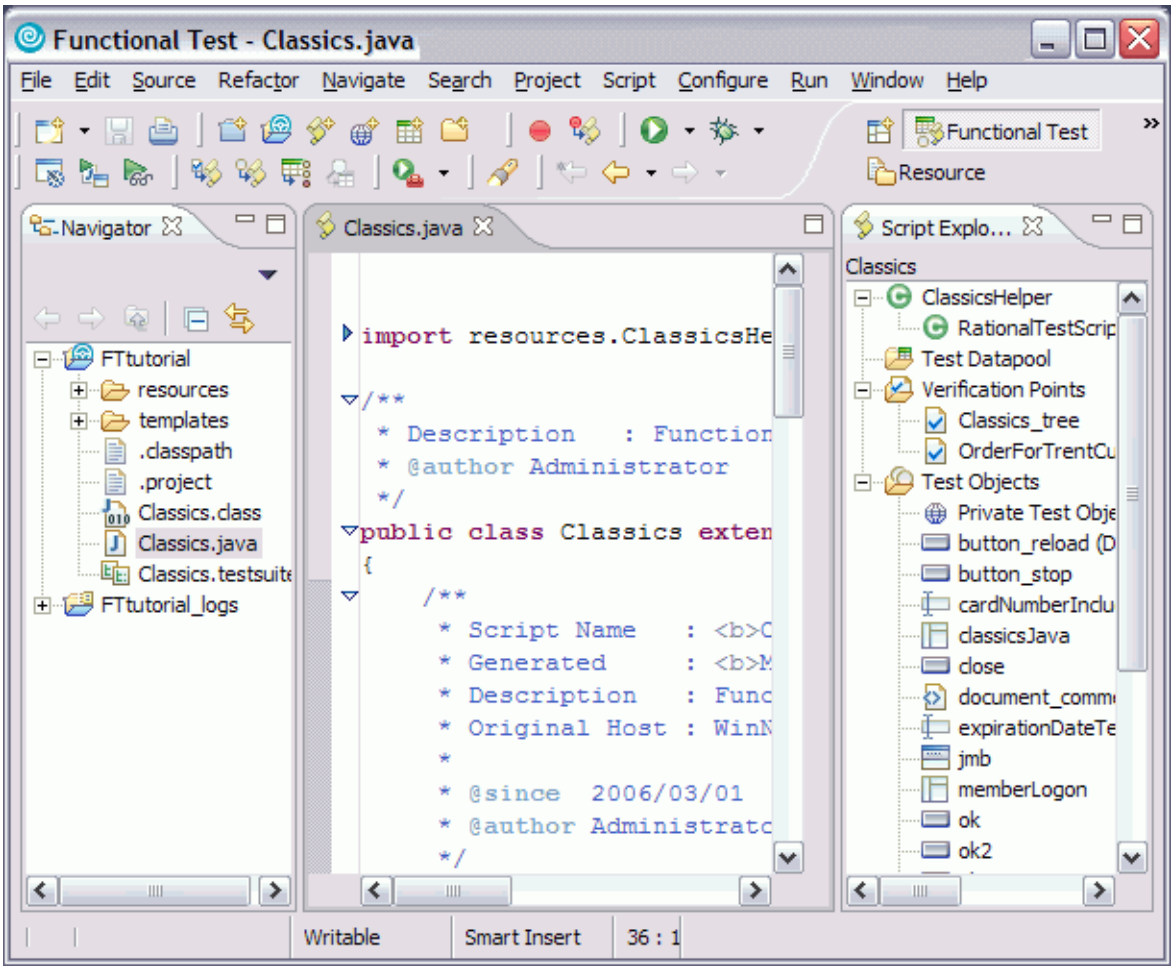
   Rational® Functional TesterHCL OneTest™ UI is minimized, and the Playback Monitor starts in the upper-right corner of your screen. As the script plays back, messages are displayed in the Playback Monitor. Rational® Functional TesterHCL OneTest™ UI plays back all of your recorded actions, such as the application starting, the actions you performed on the application, and the verification points.

   When playback is finished, the HTML log displays the results of the test run in a separate window. Each event listed in the log should include Pass in the event headings in green. Notice that the two verification points that you recorded are listed.
3. Close the log.
   Now that you have successfully recorded a script and played it back, let us look at the Functional Test perspective in more detail.
4. If the Functional Test window is minimized, restore it.
   When you have multiple scripts, Rational® Functional TesterHCL OneTest™ UI displays all open scripts in a project in the Java™ Editor (the script window).

Learn more about Java Editor: Throughout the script, notice the information about the script shown at the top in light blue and prefixed by asterisks. This information comes from the script template, which you can modify. For more information about modifying the script template, see the Rational® Functional TesterHCL OneTest™ UI Help.

Notice that Rational® Functional TesterHCL OneTest™ UI adds a short comment to the script in green characters to identify the object that the following lines refer to. This information makes it easier to navigate the script. Strings passed as arguments to methods during recording, including user inputs, are bright blue.

When your cursor hovers over certain areas of the script, Rational® Functional TesterHCL OneTest™ UI displays useful information in a pop-up text box. For example, for a helper method, you see the description property set in the object map followed by the recognition properties of the object. The hover feature is controlled by Preferences. To turn it off or modify what is shown, click **Window > Preferences**, then choose **Java > Editor** and click the **Hovers** tab. The hover feature is on by default.



To the left of the Java™ Editor (the script window) is the Functional Test Projects view, which lists any Rational® Functional TesterHCL OneTest™ UI projects to which you are currently connected. All scripts within each project are listed below the project name. This Projects view provides another way to navigate to a different script. When you double-click a script in the Projects view, it opens in the script window and becomes the active script.

To the right of the Java™ Editor is the Script Explorer, which lists the verification points and object map of the active script. From the Script Explorer, you can start the Verification Point Editor to display and edit verification points and start the object map editor to display and edit object maps. For more information about the Script Explorer or the other parts of the Functional Test perspective, such as the Tasks View and Console View, see the Rational® Functional TesterHCL OneTest™ UI Help.

Lesson 5: View verification points and object maps

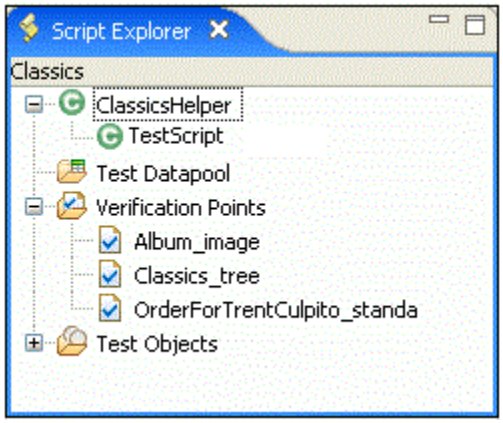
In this lesson, you will learn how to view and modify the properties of verification points and object maps.

View verification points

About this task

You can examine and modify the data inside a verification point.

1. In Rational® Functional TesterHCL OneTest™ UI, verify that your script, Classics.java, is still the active script in the Java™ Editor.
2. The three verification points you recorded should be listed in the Script Explorer to the right of the script. If necessary, click the plus sign (**+**) next to Verification Points to expand the list.



3. Double-click **Classics_tree**.

   This is the first verification point that you recorded, on the list of composers. The Verification Point Editor starts; you can update verification point data for future playbacks.

   Updating verification points: Data verification points have six possible display types. This is a Data (tree) verification point. The object type is a tree, in this case, a javax.swing.JTree. To edit the data in this tree, double-click any of the sub-items in the tree to open a small edit box where you can make changes. Use the check boxes beside each item to indicate whether you want this item to be tested in

future playbacks. To learn more about using the Verification Point Editor, see the Rational® Functional TesterHCL OneTest™ UI Help.

4. Close the Verification Point Editor.

View object maps

About this task

You can examine and modify the data inside the object map.

1. In the Script Explorer, expand the **Test Objects** folder.

   The first item, Private Test Object Map, is the object map for this script. The individual objects listed under Private Test Object Map are references to objects that were acted on during recording.

2. Double-click **Private Test Object Map** ( ) to open it.

   Object map types: When you record a script, Rational® Functional TesterHCL OneTest™ UI creates an object map for the application under test. Each script is associated with an object map file. The map file can be private -- associated with one script exclusively -- or shared among many scripts. When you recorded the script, Rational® Functional TesterHCL OneTest™ UI used the default setting (private map). The object map contains properties for each object, and you can easily update the information in one central location. Then, any scripts that reference that object also share the updated information.

3. Expand the top-level object `Java: Frame: logFrame1: javax.swing.JFrame`.

   The frame object includes the logon dialog box. The radio buttons, password fields, and action button are listed beneath the frame object.

4. Click one of the objects.

   Notice that the recognition properties are displayed in the grid below the object tree. The object map also provides a quick way to add object references to a script. In the object map menu, you can click **Test Object > Insert Object(s)** to add objects. You can also perform other operations from the object map, such as changing the weight of a recognition property and editing recognition properties and values. We'll perform several advanced procedures using the object map later in the tutorial.

5. In the object map menu, click **Preferences > Clear State On Close**.

   The **Clear State On Close** command is a toggle menu item and should be on by default, so you will be clearing it. If it were left on, all objects would be accepted when you close the map. We want to do that in a later step when we return to the object map to make changes.

6. Close the object map. Do not save any changes you may have made.

Lesson 6: Perform regression tests

In this lesson, you will execute your script on a different build. When you have a new build of an application, you can run the automated test you recorded by playing back your script on the new build. To execute your script on the new build, you must change the name of the application in your script. (You would not need to do this on a development project; you do it here to simulate getting a new build of the application.)

1. In the Java™ Editor (script window), verify that your script (Classics.java) is the active script.

   At the top of the script, beneath the template information, note the start application command:

   ```
   startApp("ClassicsJavaA");
   ```

2. Change the "A" to "B".

   Java™ code is case-sensitive, and so be sure to use an uppercase B. You do not need to save or compile the script for the change to take effect. It is done automatically when you run the script.

3. Click the **Run Functional Test Script** toolbar button  to play back the script.
4. In the Select Log window, select **Classics** and then click **Finish**.
   You will be prompted to overwrite the log.
5. Click **Yes**.
   Result

   The script begins to play back quickly, but slows near the end on the Member Logon window. That is because Build B of the application has different text in the field beside the check box. Rational® Functional TesterHCL OneTest™ UI is looking for an object that matches the recognition properties recorded in Build A. We'll show how to fix this problem later in the tutorial.

6. When the log opens after playback, look at the messages. You should see two failures and one warning in the log. (Keep the log open in preparation for lesson 7.)

   The properties verification point (OrderForTrentCulpito_standa) and the image verification point (Album_image) failed because of a change in the application. Next, we'll see how to update the verification point baseline to fix this. An object recognition warning was generated for the password check box field. We'll also show how to fix that in the object map using a regular expression in a later section of the tutorial.

   Did you notice that the main screen of ClassicsB looks different from ClassicsA? That difference did not cause the script to fail, however. The same objects are present but in a different location on the two applications. This did not cause a failure because Rational® Functional TesterHCL OneTest™ UI uses robust recognition methods to locate the objects. For example, it does not rely on superficial properties such as screen coordinates to find objects. Instead, it uses internal recognition properties. This method allows for flexibility in the user interface design, without requiring that you alter or re-record your scripts.

Lesson 7: Use the Comparator to update a verification point

You can use the Verification Point Comparator to compare verification point data after you play back a script. Verification points provide a baseline of the properties or data of an object. If the verification point fails on a

subsequent build of an application, you have found a defect or an intentional change to the application. If the change is intentional, you can update the information in the verification point so that the test continues to be valid for future builds.

Before you begin

At the end of lesson 6, you left the log open. If you closed the log, reopen it by double-clicking on the log name in the Projects view.

1. In the log, click the **View Results** link at the end of the failed image verification point entry. The event heading is "Verification Point (Album_image)."
   Result

   The Rational® Functional TesterHCL OneTest™ UI Verification Point Comparator displays your verification point data. Notice that the Comparator banner includes the name of your verification point.

   Problems with the comparator?: If the comparator does not open or you get an error message, you need to enable the Java™ plug-in of your browser. For the instructions to do that, see the topic called "Enabling the Java™ Plug-in of a Browser" in the "Before You Record" section of the Rational® Functional TesterHCL OneTest™ UI Help.

   When a verification point fails, the Comparator shows the expected and the actual values to help you analyze the differences. You can then load the baseline file and edit it or update it with the values from the actual file. Failures are displayed in red.

   When you created the verification point on ClassicsA, the captured album image is based on the object `javax.swing.JLabel`. When you played back the script on ClassicsB, since the height and the width of the object `javax.swing.JLabel` is different, the image verification point failed. So you must update the baseline file to change the object to match ClassicsB.
2. Click the **Load Baseline to Edit** button () on the Comparator toolbar.
3. Click the **Replace Baseline with actual value** button () on the Comparator toolbar.
   The actual image is loaded as the baseline image.
4. Close the Comparator.
5. In the log, click the **View Results** link at the end of the failed properties verification point entry. The event heading is "Verification Point (OrderforTrentCulpito_standard)."
6. Scroll to the **text** property.

   When you created the verification point on ClassicsA, the banner title was "Order for Trent Culpito." When you played back the script on ClassicsB, the banner title was "Orders for Trent Culpito." "Orders" is correct, because a customer might have multiple orders in the Orders window. So you must update the baseline file to change the text to match ClassicsB.

   You can only edit the baseline file.
7. Click the **Load Baseline to Edit** button () on the Comparator toolbar.

Notice that the left **Value** column displays the **Baseline Value** now.

8. Instead of scrolling to the **text** property, you can click the **Jump to First Difference** button () above the Property column. The four navigation buttons help you locate the differences between the baseline and actual files.

   You can update the baseline file in two ways. You can edit that cell of the grid, adding the letter 's' to the word "Order," or you can use the Replace Baseline command. Replacing the baseline replaces all values from the baseline file with the values from the actual file. In general, if you need to edit only one or a few values, you should edit the individual values.

9. This test has only one difference to update, so click the **Replace Baseline with actual value** button () on the Comparator toolbar.
   Both values in the **text** property now match and the property no longer appears in red. For more information about using the Comparator, see the Rational® Functional TesterHCL OneTest™ UI Help.
10. Close the Comparator.

    Now we will play back the script again to confirm the verification point passes, given the updated baseline value for the failure.

11. Close the log.
12. Click the **Run Functional Test Script** button on the Rational® Functional TesterHCL OneTest™ UI toolbar.
13. Select the **Classics** log and then click **Finish**.
14. Click **Yes** if prompted to overwrite the log.
    Result

    Rational® Functional TesterHCL OneTest™ UI pauses on the Member Logon window because you did not fix that recognition problem yet. At the end of playback, Rational® Functional TesterHCL OneTest™ UI displays the log. The verification point now passes! See how easy it is to use the Comparator to update object data and properties to account for changes in the application under test.

15. Leave the log open.

Lesson 8: Update the object map

In this lesson, you will fix the object recognition warning by using the object map. You will also use a regular expression for more flexible object recognition.

About this task

When you see a recognition failure or warning, look at the log message. At the end of lesson 7, you left the log open. If it is not open, open it by double-clicking the log in the Projects view. One warning remains in the log. The event heading is **Object Recognition is weak (above the warning threshold)**.

1. Look at the **ObjectLookedFor** and **objectFound** fields in the warning section near the bottom of the log.

   In ClassicsA, the name of the password field is **Remember Password**. In ClassicsB it is **Remember The Password**. When you played back the script on ClassicsB, the object recognition did not match exactly because of this difference.

2. Look at the **Line Number** field in the log. Note the number and close the log to return to Rational® Functional TesterHCL OneTest™ UI.
3. Click anywhere in the script window, and then click **Navigate > Go to Line**.
4. Type the line number from the log failure message, and then click **OK**.
   Result

   The cursor moves to the left margin of that line number.

   Note: You can also find the line number by looking at the indicator in the bottom of the Rational® Functional TesterHCL OneTest™ UI window. For example: "43:9" refers to position 9 on line 43.

   The line in your script should be:

   ```
   RememberPassword().clickToState(SELECTED);
   ```

   This line represents your click action on the password check box. This line in the script shows which object is failing. Now you can look for that object in the object map.
5. To find the object, return to the list of Test Objects in the Script Explorer (right pane).
   You should see `rememberPassword` listed under the **Test Objects** folder.

View the object recognition properties in the object map

1. Double-click the **rememberPassword** object to open it in the object map.
2. Click **Test Object > Accept All** on the object map menu. If the command is grayed out, don't do anything.
   Result

   Notice that all the objects change to black text. The text is blue (to indicate new objects) until you accept the objects in a map. You should accept the objects the first time you look at a newly created object map.

3. If the password check box object is not selected in the map, select it. (It is the object called **Java: checkBox: checkRemember: javax.swing.JCheckBox**.)
4. Look at the recognition properties listed in the **Recognition** tab at the bottom of the object map.
   Result

   You can see that this is the object from ClassicsA, because it says `Remember Password` in the **text** property. This is the "old" object. However, when you played back the script on ClassicsB, the text for that object changed, so Rational® Functional TesterHCL OneTest™ UI recognizes it as a "new" object. You want to use the new object properties in this case, so you must add it to the map.

Add the new object to the map

About this task

To add the new object to the map, open ClassicsB and then open the Member Logon window.

1. Click **Applications > Run** in the object map menu.
2. Select **ClassicsJavaB**. (Be sure to pick B).
3. Click **OK**.
4. In ClassicsCD, select any CD and then click **Place Order**.
   Result

   The Member Logon window opens.
5. Move the object map lower on your screen, if necessary, to see all of it. In the object map menu, click **Test Object > Insert Object(s)**.

   This is the same as the Object Finder tool in the Select an Object page of the Verification Point Wizard.
6. Clear the **After selecting an object advance to next page** check box if it is selected.
7. Use the Object Finder tool to select the **Remember the Password** check box in the Member Logon window.

   After you select the check box, you'll see that the **text** property is now `Remember The Password`. Stretch the borders of the object map, if necessary, to see the properties.
8. On the Select an Object page, click **Next**.
9. Don't change anything on the Select Object Options page, and then click **Finish**.
   Result

   The new check box object is now shown in the object map.
10. Click another object and notice that the new item is listed in blue and the word "New" is displayed at the beginning of the line.
    Result

    Now both the old and the new objects are listed in the map. You want to unify the two objects and take the properties from each that you want for the new object.

Unify the objects

1. To unify the objects, click the old object (the original check box labeled **CheckBox: checkRemember**) and then drag it onto the new object in the list. Position the tip of the cursor arrow over the new object before you release the mouse button. Then, release the mouse button.
   Result

   The Unify Test Objects wizard opens.
2. Widen the Unify wizard if necessary to see more of the information in the lower sections.
   Result

In the lower left section, the original object's properties are shown. It should be labeled "Source: RememberPassword." That is what the text was on the check box in ClassicsA. In the lower right section, it should be labeled "Target: RememberThePassword." That is what the text is on the check box in ClassicsB.

Because you dragged the old object to the new object, the new object's recognition properties are filled in at the top of the wizard. In general, Rational® Functional TesterHCL OneTest™ UI puts the new properties at the top if they are the preferred properties. However, some old administrative properties might be preferred. For example, Rational® Functional TesterHCL OneTest™ UI retains regular expressions in the old property set. To use a property from the old object, double-click that property in the grid of the old object and it will be copied up into the unified object. In this case, we want to use all the properties of the new object, which are already filled in.

3. Click **Next**.
   Result

   All scripts that are affected by this change in the object map are listed. Only one script, Classics, is affected.
4. Click **Finish**.
5. In the object map, click the **File > Save** menu on the object map toolbar to save the changes you made and then close the object map.

Play back the script again

About this task

Now we'll play back the script again on ClassicsB to confirm that it passes.

1. Close both dialog boxes of ClassicsCD.
2. In Rational® Functional TesterHCL OneTest™ UI, click **Run Functional Test Script** on the toolbar.
3. Select the **Classics log** and then click **Finish**.
   Result

   The script now passes with no warnings! Notice that the playback no longer pauses on the password check box object because the recognition properties now match.

   This object unification feature is an easy way to update scripts when recognition properties of an object intentionally change. One of the major advantages of this feature is that if your object map is being used by many scripts, you could update them all when you make the change in the wizard. Instead of manually editing multiple scripts, you can make a change once in the map and the change propagates automatically to all scripts that use it. This feature can save you time.

   Another way to update recognition properties: There is also an easier way to update the recognition properties of a test object should they change. Instead of using the Unify wizard as described in this exercise, from the Object Map you can select the test object whose recognition properties you want to update. Right-click the test object as it is displayed in the Object Map tree and select **Update Recognition Properties** from the pop-up menu. You will need to have the test application running

when this action is performed so that Rational® Functional TesterHCL OneTest™ UI can get the updated recognition properties. You would only use this update method if you do not want to use any properties of the old object.

4. Close the log.

Lesson 9: Change the Recognition Preferences

In the previous lesson, you saw how you can update the recognition properties of an object when they change. Another factor you can change is the recognition weights that Rational® Functional TesterHCL OneTest™ UI uses during playback. You use the ScriptAssure™ recognition preferences to set this. The label object that you tested with the second verification point can demonstrate how this works.

1. On the Rational® Functional TesterHCL OneTest™ UI menu, click **Window > Preferences**.
2. Click **Functional Test > Playback > ScriptAssure**.
3. Click the **Advanced** button.

   Notice that one of the default settings is **Warn if accepted score is greater than: 10000**. A score of 10000 indicates that one important property can be wrong. Let's lower the score to 5000 and see what happens.
4. Select the **Use Default** check box beside this field.
5. Then type `4000` in the field and then click **OK**.
6. Play back the script on ClassicsB again.
   Result

   The log now contains a warning for the label object. The reason given in the **objectFound** field, is that the recognition score is 10000. This discrepancy was caused by changing the word "Order" to "Orders" in the label.
7. Close the log.
8. Restore the default value for the recognition score:

   a. Click **Window > Preferences**.

   b. Click **Functional Test > Playback > ScriptAssure**.

   c. Click the **Advanced** button.

   d. Select the **Use Default** check box beside the **Warn if accepted score . . .** field.
      Result

      This will change the 4000 back to 10000.

   e. Click **OK**

   f. Play back the script again.
      Result

Now the warning is gone and everything passes.

g. Close the log.

Results

This lesson showed how you can tweak the recognition score in order to achieve the sensitivity that you want for object recognition. For more information about using ScriptAssure™, see the Rational® Functional TesterHCL OneTest™ UI Help.

Lesson 10: Use regular expressions

The last thing you will do using the object map is convert a property value to a regular expression. In this case, the regular expression provides more flexibility in the object recognition.

About this task

We just saw how the script passes completely on ClassicsB. That was the goal because the changes made to the application in ClassicsB are correct. So the script is now in the state you want it to be in going forward. Now when you play the script back against ClassicsA, it fails because of the changes made earlier. You might want to allow more than one variant of an object to pass. You might have a dynamic object or have several versions of your application with slightly different versions of an object, in which both are correct. You can use a regular expression to allow more than one version of a property value, such as text, to accommodate this scenario.

Open the object map and unify the objects

1. To play back against ClassicsA, edit the startApp command at the top of the script and change the B to an A.
2. Click **Run Functional Test Script** on the Functional Test toolbar.
   During playback, Rational® Functional TesterHCL OneTest™ UI pauses a little on the password check box object, but eventually it finishes. The script now gives a warning. Notice in the log that it's the same object, the **rememberPassword** test object.
3. Close the log and then open the object map from the password check box object as you did in Lesson 8, by double-clicking **rememberPassword** in the Script Explorer.
4. In the object map, open the application by clicking **Applications > Run**. Select **ClassicsJavaA** and then click **OK**.
5. Pick any CD and click **Place Order** in ClassicsCD to open the Member Logon window.
6. Add the new object to the map by clicking **Test Object > Insert Object(s)**.
7. Use the Object Finder to select the password check box in the Member Logon window in the application.
8. Click **Next**, and then click **Finish**.
9. In the top pane of the object map, drag the old check box object to the new check box object to unify the objects.
10. Widen the Unify Test Objects wizard by dragging one of the sides outward to make the fields longer, if necessary.

You will use two different regular expressions: one on the **name** property and one on the **text** property.

The unified object is shown in the **Unified Test Object Properties** grid (top pane); the **name** property has a value of `checkRemember`.

Convert a property value to a regular expression

1. In the top pane, right-click the `checkRemember` value and then click **Convert Value to Regular Expression**.
   Result

   Rational® Functional TesterHCL OneTest™ UI designates the value as a regular expression by the "xy" icon in front of the value text.
2. Double-click the **name** value again so that you can edit the field.
3. Delete the word `check` and then edit the remainder to read: `[rR]emember`.
4. Click outside the cell.

   This pattern allows the word "remember" with either an uppercase "R" or lowercase "r" to pass. This is important because the comparisons are case-sensitive, and only an exact match will pass. The value of the **text** property is "Remember Password".
5. Right-click the Remember Password value and then select **Convert Value to Regular Expression** to convert it.
6. Double-click the value and edit it to read: `Remember.*Password`. You are removing the space and adding the period (.) and asterisk (*) characters.
7. Click another cell.

   The "**.**" allows any character to appear in that position. In one version of the application, there is a space between the two words in this property, and in the other version there is no space. This pattern covers both cases.
8. Click **Next**, and then click **Finish**.
9. Click **File > Save** in the object map to save the changes, and then close the object map.
10. Close ClassicsCD.
11. Play back the script again on ClassicsA. The image verification point and the properties verification point fails.
    Result
    The image verification point fails because the height and the weight of the object`javax.swing.JLabel` is different. The properties verification point is expected to fail because the text `Orders for Trent Culpito` was never changed to a regular expression. The object recognition warning on ClassicsA is no longer in the log.
12. Close the log.
13. Change the startApp command to play back ClassicsB, and then run the script.
    Result

    The object recognition also passes on ClassicsB Regular expressions offer more flexible recognition for an object that has different properties in different versions of an application, and both are recognized during playback. For more information about regular expressions, see the Rational® Functional TesterHCL OneTest™ UI Help.

Summary: Create functional tests

This tutorial has shown you how to set up Rational® Functional TesterHCL OneTest™ UI for testing, recording and playing back scripts, creating verification points and using the Verification Point Comparator to update object properties or data, and several ways to use the object map to your advantage.

Lessons learned

By completing this tutorial, you learned how to:

- Create a Functional Test project
- Record a script against actions on your test application
- Start your test application properly while recording
- Create verification points
- Play back scripts
- Use the Functional Test log
- Update verification points using the Comparator
- Update the object map
- Change recognition preferences for an object
- Use regular expressions for more flexibility in object recognition

Additional resources

If you want to learn more about the topics covered in this tutorial, consult the following resources:

- Product Help
- API Reference
- Welcome Page

Perform a data-driven functional test using Java scripts

In this tutorial, you will learn how to create a data-driven functional test using the Rational® Functional TesterHCL OneTest™ UI data-driven test wizard.

Data-driven testing puts a layer of abstraction between the data and the test script, eliminating literal values in the test script. Because data is in a dataset and separated from the test script, you can:

- Modify test data without affecting the test script
- Add new test cases by modifying the data, not the test script
- Share the test data with many test scripts

Learning objectives

After completing this tutorial, you will be able to:

- Create a project and record a Java test script
- Data-drive a test

- Add descriptive headings to the data
- Create a verification point with a dataset reference
- Add data to the dataset
- Play back the test

Time required

30 minutes.

Related information

Create a functional test using Java scripts on page 104

Introduction: Perform a data-driven functional test using Java scripts

In this tutorial, you will learn how to create a data-driven test using a variety of realistic data to test the application with the Rational® Functional TesterHCL OneTest™ UI data-driver wizard.

You will use the ClassicsCD sample application to create a project and record a Java test script to verify that the ClassicsCD sample application correctly totals an order. You will also create a verification point with a dataset reference to check that the total amount of the order is correct in the Classics CD application.

Learn more about datasets: A dataset is a collection of related data records. A dataset supplies data values to the variables in a test script during test script playback. Data-driven testing uses data from an external file, a dataset, as input to a test.

The diagram on the left shows a test script that uses data with hard-coded, literal references in the test script. The diagram on the right shows a data-driven test script that uses data from an external file, a dataset.

Hard-coded test script	Data-driven test script



Learning objectives

After completing this tutorial, you will be able to:

- Create a project and record a Java test script
- Data-drive a test
- Add descriptive headings to the data
- Create a verification point with a dataset reference
- Add data to the dataset
- Play back the test

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Lesson 1: Create a project and record a test script

In this lesson, you will use the Classics CD sample application to create a new project and start recording a test to verify that the sample application correctly totals the amount of music CDs purchased.

About this task

What is a project?: A project is a collection of test assets such as test scripts, object maps, verification points, and datasets, that can facilitate the testing of one or more software components. You must create a functional test project before you can record a test.

Disable simplified scripting and application visuals feature

About this task

Rational® Functional TesterHCL OneTest™ UI provides you the option to generate simplified test scripts and Java test scripts. If you are familiar with Java scripting, you can disable the simplified scripting and application visuals feature and start recording the test script. In this tutorial, we will work with Java test scripts. Before you start recording the scripts, disable the simplified scripting and application visual features.

1. To verify whether the feature is enabled, click **Window > Preferences**.
2. In the left pane of the **Preferences** window, expand **Functional Test**, then **Simplified scripting**.
3. In the **Simplified Scripting** page, clear the **Enable Simplified Scripting** checkbox.
4. In the **Application Visuals** page, clear all the options listed in the page for the application visuals.
5. Click **Apply** and then **OK**.

Create a project

About this task

Create a project to store the test assets that you need to test the Classics CD sample application.

1. Click **Windows > Open Perspective > Other** to open the functional test perspective . In the Open Perspective dialog box, select the **Functional Test** option.
2. Click **File > New > Functional Test Project**.
3. Type `DataDriveTutorial` for the name of the new project.
4. Click **Finish**.

Start recording

About this task

Start recording a test script to verify that when a customer orders a music CD, the total amount charged to the credit card is the correct amount listed in the application.

1. On the Functional Test toolbar, click **Record a Functional Test Script**  .
2. Type `OrderTotal` for the name of the test script.
3. Click **Next**.
   Result

   When you create a test script, Rational® Functional TesterHCL OneTest™ UI creates a test dataset and other test assets. Use the defaults for **Private Test dataset** and **Sequential**. A private test dataset is associated with only one script and is not available to any other scripts. When you use the sequential order, the test script accesses dataset records in the order that they appear in the dataset.
4. Click **Finish**.
   Result
   The Rational® Functional TesterHCL OneTest™ UI window minimizes and the Recording Monitor opens.

Start the ClassicsCD application

About this task

Start the ClassicsCD application and navigate through the application to the dialog box that you will data-drive.

1. On the Recording toolbar, click **Start Application**().
2. If necessary, click the **Application Name** arrow to see the options, and then select **ClassicsJavaA - java**.
3. Click **OK**.
   ClassicsJavaA is build 1 of the sample application, ClassicsCD, which comes with Rational® Functional TesterHCL OneTest™ UI.
4. In the ClassicsCD application, under **Composers**, double-click **Schubert** to open the list of CDs for sale by that composer, and then click **String Quartets Nos. 4 & 14**.
5. Click **Place Order**.
6. Click **OK** to close the Member Logon window.
7. In the Place an Order window, type `1234567890` in the **Card Number** field and then type `09/09` in the **Expiration Date** field.
   Result

   Proceed to Lesson 2. We will data-drive a test by populating the data from this dialog box.

Lesson 2: Data-drive a test

In this lesson, you will use the data-driver to populate a dataset with data from the sample application. A dataset is a collection of related data records. A dataset supplies data values to the variables in a test script during test script playback.

1. On the Recording toolbar, click **Insert Data Driven Commands**  .
   Result
   The recording pauses.
2. In the Insert Data Driven Actions page, use the mouse to drag the Object Finder () to the title bar of the **Place an Order** window on the **ClassicsCD** application.
   Result
   Rational® Functional TesterHCL OneTest™ UI outlines the entire Place an Order window with a red border.
3. Release the mouse button.
   Result
   In the Data Drive Actions page, under the **DataDriven** Commands table, information about the selected objects are displayed.

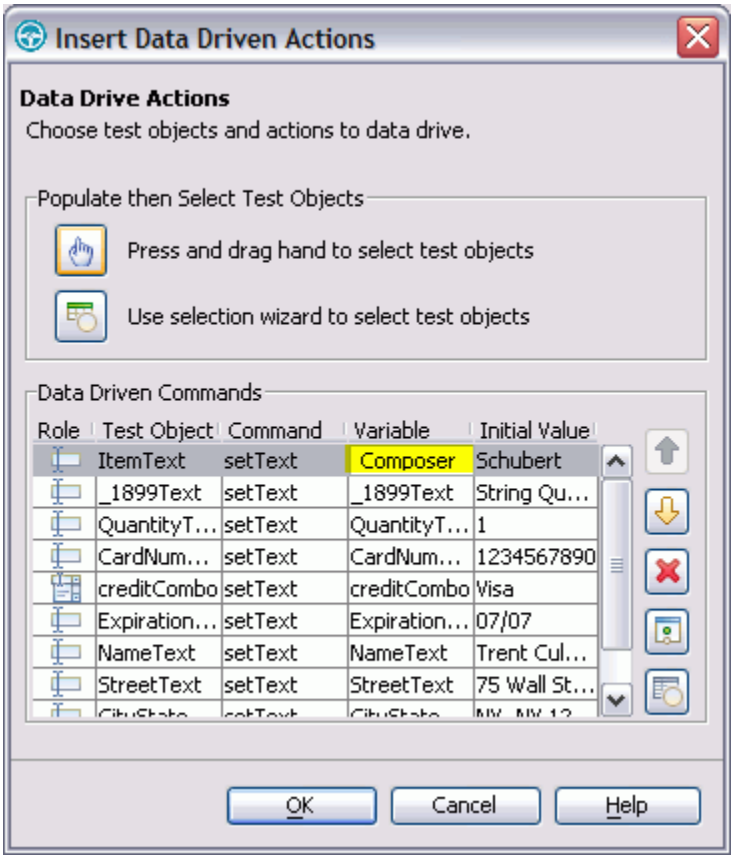
Results

You can hover over a row in this table to view the line of code that Rational® Functional TesterHCL OneTest™ UI inserts into the test script to data-drive the test script.

Lesson 3: Add descriptive headings to the data

In this lesson, you will add descriptive headings to the dataset you created in the previous lesson. Descriptive headings make it easier to add data to the dataset.

1. In the **Data Driven Commands** table, under the **Variable** header, replace **ItemText** with `Composer`.



2. Repeat sequentially, replacing each cell in the **Variable** column with a descriptive name for each heading in the **Variable** field. Use the text in the following variables list as descriptive names.

Note: Do not use spaces in **Variable** names. Typically, you would look at the application to determine the appropriate headings for each row, but we have done that for you in the following variables list:

Variable
Composer
Item
Quantity
CardNo
CardType
ExpiryDate
Name
Street

Variable

CityStateZip

Phone

Rational® Functional TesterHCL OneTest™ UI automatically updates the test script as you change each of the **Variable** names.

3. Click **OK**.

Results

Now the dataset has descriptive headings that make it easier to add more data. You will add more data to the dataset after you finish recording the test script.

Lesson 4: Create a verification point with a dataset reference

In this lesson, you create a verification point with a dataset reference to check that the price for the CD is correct in the ClassicsCD application.

About this task

What is a verification point?: A verification point captures object information and literal values from the application-under-test and stores it as the baseline for comparison during playback. When you play back the script, a verification point captures the object information again to compare it to the baseline and see if any changes have occurred, either intentionally or unintentionally. Comparing the actual object information in a script to the baseline is useful for identifying potential defects.

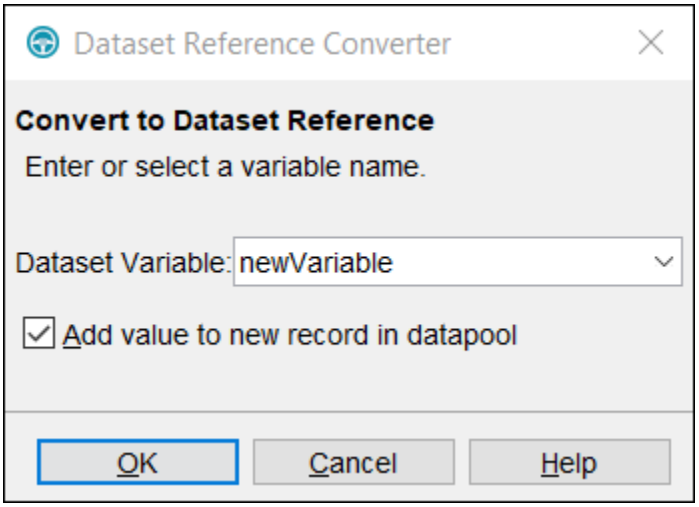
You will use a dataset reference instead of a literal value for the value that you are testing in the verification point. Using datasets with verification points gives you more flexibility to test realistic data with your test scripts.

Create a verification point with a dataset reference

1. On the **Recording** toolbar, click **Insert Verification Point or Action Command** .
   Result
2. In the Verification Point and Action Wizard, use the mouse to drag the **Object Finder** to $18.99, which is next to "Sub-Total" in the Classics CD application.
   Result
   Rational® Functional TesterHCL OneTest™ UI outlines $18.99 with a red border.
3. If the Select an Action page is not displayed, click **Next**.
4. In the Select an Action page, click **Perform Data Verification Point** to test whether the price of the CD changes.
5. Click **Next**.
6. In the Insert Verification Point Data Command page, click **Next**.

7. On the Verification Point Data page toolbar, click **Convert Value to dataset Reference** ( ) to use a dataset instead of a literal value in a verification point. (If you cannot see the **Convert Value to dataset Reference** button on the toolbar, make the page larger by dragging a corner of the page).
   Result

   The dataset Reference Converter dialog box opens.



8. In the dataset Variable field, type `Price` to replace the **newVariable** for the heading in the dataset.
9. Select the **Add value to new record in dataset** check box to add the **Price** to the existing dataset record you created in the previous exercise.
10. Click **OK**.
11. Click **Finish**.

Place the order and close the ClassicsCD application

1. In the **ClassicsCD** application, click **Place Order** to place the order, and then click **OK** to close the message confirming your order.
2. Click **X** in the upper right corner of the **Classics CD** application to close the application.

Stop recording

On the **Recording** toolbar, click **Stop Recording** ( ) to write all recorded information to the test script.

Results

The test script is displayed in the editor window.

Lesson 5: Add data to the dataset

In this lesson, you will add data to the dataset to test the ClassicsCD sample application by placing more orders for the CD.

1. In the Script Explorer, double-click **Test dataset** and then double-click **Private Test dataset**. In the test script editor, double-click the **Test dataset** tab to expand the dataset editor so that you can work.
   Result

   The dataset editor opens and should look similar to the following table:

	Composer	Item	Quantity	Card#	CardType	ExpDate	Name	Street	CityStZip	Phone	Price
0	Schubert	String Quartets Nos. 4 & 14	1	1234567890	Visa	09/09	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$18.99

2. Position your mouse pointer in the dataset editor, then click **Enter** to add a row after the first row.
3. To add a second empty row, right-click **Insert Record**.
   To save time, copy the data from row 0 in the dataset into the two empty rows that you created in steps 2 and 3.
4. Position the mouse pointer in the row 0 cell, right-click, and then click **Copy**.
5. Position the mouse pointer in the row 1 cell, right-click, and then click **Paste**.
6. Click **Yes** to paste the data into the empty row.
7. Position the mouse pointer in the row 2 cell, right-click, and then click **Paste**.
8. Click **Yes** to paste the data into the empty row.
9. Change the value in the **Quantity** column to test the ClassicsCD sample application by placing more orders for the CD:
   a. In row 1, in the Quantity column, select the cell and type 2.
   b. In row 2, in the Quantity column, select the cell and type 3.

   Result
   The data in the dataset should look like the following table:

	Composer	Item	Quantity	Card#	CardType	ExpDate	Name	Street	CityStZip	Phone	Price
0	Schubert	String Quartets Nos. 4 & 14	1	1234567890	Visa	09/09	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$18.99
1	Schubert	String Quartets Nos. 4 & 14	2	1234567890	Visa	09/09	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$18.99
2	Schubert	String Quartets Nos. 4 & 14	3	1234567890	Visa	09/09	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$18.99

10. On the **Test dataset** tab, click **X** to close the dataset editor, and then click **Yes** to save the changes you made to the dataset.

Lesson 6: Play back the test

In this lesson, you will play back the test you just recorded to see how easy it is to use a variety of data from a dataset to test the application.

About this task

Each time you play back a script with an associated dataset, the script accesses one record in the dataset. When you create a dataset reference for a verification point, the verification point uses the dataset reference to access a variable in that record. During playback, Rational® Functional TesterHCL OneTest™ UI substitutes the variable in the dataset for the dataset reference and compares the variable in the dataset to the actual results.

During playback you can view the script name, the script line number that is executing, status icons, and a description of the action in progress in the Playback Monitor.

1. To play back the test script, click **Script > Run**.
2. In the Select log window, click **Next**.
3. Click the **dataset Iteration Count** arrow and then scroll to select **Iterate Until Done** to access all three records in the dataset.
4. Click **Finish** to use the default log name.
   Result

   The Rational® Functional TesterHCL OneTest™ UI window minimizes, and the Playback Monitor is displayed in the upper-right area of your screen. Messages appear in the Playback Monitor as Rational® Functional TesterHCL OneTest™ UI plays back all of the recorded actions in the test script and enters data from the dataset.

   When the test script finishes playing back, Rational® Functional TesterHCL OneTest™ UI displays a log with the test results. A log is a file that contains the record of events that occur while playing back a script. A log includes the results of all verification points executed that can be used to test the application.
5. Click **X** to close the log.

Summary: Create a data-driven test

This tutorial has shown you how to create a data-driven test.

You have created a data-driven test script, created descriptive headings for the data collected, added data to the dataset, created a data verification point with a dataset reference, played back a test script, and viewed the log.

Lessons learned

By completing this tutorial, you learned how to:

- Create a project and record a test script
- Data-drive a test
- Add descriptive headings to the data
- Create a verification point with a dataset reference

- Add data to the dataset
- Play back the test

Additional resources

If you want to learn more about the topics covered in this tutorial, see the Data-Driving Tests section of the Rational® Functional TesterHCL OneTest™ UI Help.

Test Adobe Flex application

This tutorial walks you through the steps to enable your Adobe Flex application, and test the enabled Flex application from a local test computer using Rational® Functional TesterHCL OneTest™ UI. The steps are based on the roles that Flex developers and testers perform.

Rational® Functional TesterHCL OneTest™ UI supports testing the functional aspects of Flex applications. You can record and playback scripts against Flex based user interfaces inside a web browser.

Learning objectives

After completing this tutorial, you will be able to use Rational® Functional TesterHCL OneTest™ UI to test your Flex applications.

Time required

30 minutes

Related information

Tutorial: Create a functional test on page 104

Tutorial: Create a data-driven functional test on page 124

Introduction: Test Adobe Flex application

In this tutorial, you will learn how to enable a Flex application for functional testing and test the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI.

The tutorial is divided into two modules that must be completed in sequence to work properly.

Learning objectives

After completing this tutorial, you will be able to:

- Set up the development and testing environment
- Enable the Flex application for functional testing
- Test the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI

Time required

This tutorial takes approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Module 1: Enable the Flex application for testing

In this module, you learn how to enable the Flex application for functional testing by compiling the Flex 2.0 application with functional testing agent (rft.swc) and Flex automation framework libraries.

This module is intended for Flex application developers.

Learning objectives

After you complete the lessons in this module you will understand how to do the following:

- Set up the development environment
- Configure the Flex application

Time required

This module takes approximately 15 minutes to complete.

Prerequisites

Before you begin, verify that the following software is installed in your computer:

- Adobe Flex SDK 4.5
- Adobe Flex automation framework

Lesson 1: Configure the Flex application

In this lesson, you use the Flex user interface to setup the configuration parameters for the Flex application.

Before you begin

You must have read and completed the actions related to the development environment. See Setting up the development environment for Flex applications on page 811.

1. Click **Configure > Configure Applications for Testing**.
2. Click **Add** in the Application Configuration Tool window.
3. Select **Flex Application**, and click **Next**.
4. Select **Configure Flex application setup**, and click **Next**.
5. Click **Local Application**.
6. Specify the Flex application parameters.
    a. Select the required Flex version from **Flex SDKs** list.
    b. Select **Compile-time** from the Enablement type list.
    c. Click **Browse** to select the Flex application in .as or .mxml format.
    d. Select **Dependency Files** check box, if required.

e. Click **Add** and browse to the dependency files.
f. Select **Additional Libraries** check box.
g. Click **Add** and browse to the libraries.
h. Click **Browse** and select the SWF target location.
i. Select the **Generate HTML Page** checkbox, and click **Finish**.

Results

The selected Flex application and its detailed information is displayed in the Application Configuration Tool window. Pass the generated HTML page to the tester to test the HTML page using Rational® Functional TesterHCL OneTest™ UI.

Module 2: Test the Flex application

In this module, you learn how to test the functionality of the enabled Flex application that is provided by the developer.

This module is intended for testers and describes the steps for testing the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI.

Learning objectives

After you complete the lessons in this module you will understand how to do the following tasks:

- Set up the testing environment
- Test the enabled Flex application from a local test computer

Time required

This module takes approximately 15 minutes to complete.

Prerequisites

Before you begin, verify that the following software is installed:

- Rational® Functional TesterHCL OneTest™ UI
- Supported browser
- Adobe Flash Player ActiveX control version 9.0.28.0 or later

Lesson 1: Assign trust designations

In this lesson, you assign trust designations to enable the file to be trusted.

About this task

To test the application from a local test computer, the application-under-test has to be a trusted application. The paths to individual files or directories can be assigned trust designations. This renders that all the files in each selected directory and any of its subdirectories are trusted.

1. Create a folder named FlashPlayerTrust in `C:\WINDOWS\system32\Macromed\Flash`.
2. Create a file named Flex without any file extension.
3. Enter the path of the Flex application in the Flex file.
4. Save the file.

Lesson 2: Security settings for Flex applications

In this lesson, you specify security settings for Flex applications.

1. Open the Flex application in Flash Player, in your browser.
2. Right-click the application and select **Settings** to access **Settings Manager**.
3. Select the **Privacy** tab.
4. Click **Advanced**.

   Result

   Adobe Flash Player launches a new browser window and loads the Settings Manager help page.
5. Click **Global Security Settings** panel link.

   Result

   The **Global Security Settings** window opens.
6. Add your application directory into secured or trusted directory. In the **Always trust files in these locations** drop down menu, click **Add location**. Browse for the location.

Note: For more information about setting the security configuration, see the Adobe® website.

Lesson 3: Test enabled Flex application from a local test computer

In this lesson, you test the functionality of the enabled Flex application that is embedded in an HTML wrapper on a local computer.

About this task

To test the enabled Flex application on a local computer:

1. Get the HTML wrapper from the developer.
2. Open the HTML page in a web browser.
3. Start Rational® Functional TesterHCL OneTest™ UI for testing the HTML application that contains the embedded Flex application.

Note: You can also test the enabled Flex application that are deployed on a web server. For more information, see the online help.

Summary: Test Adobe Flex application

This tutorial has shown you how to enable a Flex application for functional testing, and test the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI.

Lessons learned

By completing this tutorial, you learned how to perform the following tasks:

- Identify the roles played by the Flex Application developer and tester for successful automation of Flex applications
- Set up the development and testing environment
- Enable the Flex application for testing by compiling the Flex application with functional testing agent and Flex automation framework libraries.
- Test the enabled Flex application from a local test computer

Additional resources

If you want to learn more about the topics that are covered in this tutorial, consult the following resources:

- Rational® Functional TesterHCL OneTest™ UI Help
- Rational® Functional TesterHCL OneTest™ UI Welcome Page

Test GEF applications

The movies in this tutorial show you how to use Rational® Functional TesterHCL OneTest™ UI to test applications that are based on the Graphical Editing Framework (GEF). It also shows how Rational® Functional TesterHCL OneTest™ UI works with GEF objects.

Learning objectives

This tutorial is divided into two modules and each module is divided into lessons. After an overview of the GEF objects in the application under test, you will watch these tasks being performed:

- Enabling GEF applications.
- Testing the function of GEF objects using Rational® Functional TesterHCL OneTest™ UI.
- Creating data, image and properties verification points on the GEF objects, given that Rational® Functional TesterHCL OneTest™ UI recognizes the GEF edit parts and palettes.
- Identifying GEF test objects using the scroll logic.
- Identifying GEF palette objects.
- Configuring object recognition properties.

Time required

This tutorial requires approximately 45 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Introduction: Test GEF applications

This tutorial is designed for Rational® Functional TesterHCL OneTest™ UI to test the function of Graphical Editing Framework (GEF) objects that are implemented using GEF. For this tutorial, GEF objects were created using IBM®

Rational® Systems Developer Version 7.0.5 (the application under test). The Rational® Functional TesterHCL OneTest™ UI test object map lists the GEF objects in the application under test. The test object map contains recognition properties for each GEF object.

Learning objectives

This tutorial is divided into two modules and each module is divided into lessons. You will see how to test GEF applications using Rational® Functional TesterHCL OneTest™ UI.

Time required

This tutorial requires approximately 45 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Skill level

Intermediate and advanced

Audience

This tutorial is divided into two modules.

1. Module 1 provides an overview of the GEF objects in the application under test and shows you how to enable the GEF application.
2. Module 2 shows how to extend Rational® Functional TesterHCL OneTest™ UI capabilities to test the GEF applications.

Module 1: Test GEF applications

This module will give you an overview of the Graphical Editing Framework (GEF) objects that are created in IBM Rational Systems Developer, which is the application under test. You will also see how to enable GEF applications and create verification points to test the GEF objects. In this module you will observe how Rational® Functional TesterHCL OneTest™ UI recognizes the GEF edit parts in the application under test.

Learning objectives

After you complete the lessons in this module, you will have an overview of GEF objects in the application under test and see these tasks performed:

- Enabling GEF applications.
- Testing the function of GEF objects by using Rational® Functional TesterHCL OneTest™ UI.
- Creating data, image and properties verification points on the GEF objects, given that Rational® Functional TesterHCL OneTest™ UI recognizes the GEF edit parts and palettes.

Time required

This module requires approximately 25 minutes to finish.

Lesson 1: Overview of GEF objects in the application under test

Rational® Functional TesterHCL OneTest™ UI can now recognize objects in the application under test that supports GEF applications. For this tutorial, we are using IBM Rational Systems Developer Version 7.0.5.3 that supports Graphical Editing Framework (GEF) as the application under test. In Rational Systems Developer, we have inserted class diagrams that represents log files using the palette. Using Rational® Functional TesterHCL OneTest™ UI, we will test the functionality of these class diagrams. In this lesson, you will have an overview of the GEF objects that are inserted in Rational Systems Developer.

About this task

Show Me

Lesson 2: Enabling a GEF application

Before testing GEF applications using Rational® Functional TesterHCL OneTest™ UI, you must enable the application under test for functional testing. In this lesson, you will see how to enable the test applications that supports Graphical Editing Framework for functional testing.

About this task

Show Me

Lesson 3: Recording a functional test script

In this lesson, you will see how to record a functional test script to test the functionality of the Graphical Editing Framework (GEF) application. You will watch the recording behavior of Rational® Functional TesterHCL OneTest™ UI as data, image, and properties verification points are created on the GEF objects and displayed in the functional test script.

About this task

Show Me

Module 2: Applying Rational® Functional TesterHCL OneTest™ UI capabilities to GEF objects

In this module, you see how Rational® Functional TesterHCL OneTest™ UI capabilities are applied to Graphical Editing Framework (GEF) objects. Rational® Functional TesterHCL OneTest™ UI identifies GEF objects that are not visible in the work area. It also recognizes GEF palettes in the application under test. You will see how to make the functional test scripts more resilient to changes by configuring object recognition properties.

Learning objectives

After you complete the lessons in this module, you will understand how to perform these tasks:

- Identifying GEF test objects using the scroll logic.
- Identifying GEF palette objects.
- Configuring object recognition properties.

Time required

This module requires approximately 20 minutes to complete.

Lesson 1: Identifying GEF test objects using the scroll logic

In this lesson, you will observe how Rational® Functional TesterHCL OneTest™ UI identifies Graphical Editing Framework (GEF) test objects that are not displayed in the IBM Rational Systems Developer work area. If the test objects are moved beyond the work area after recording the script, Rational® Functional TesterHCL OneTest™ UI scrolls beyond the work area and identifies the GEF test objects during playback. For example: When you change the resolution of the screen from 1024 x 720 to 800 x 600, the test objects in the work area is hidden. Rational® Functional TesterHCL OneTest™ UI scrolls beyond the work area and identifies these test objects during playback.

About this task

Show Me

Lesson 2: Identifying GEF palette objects

In this lesson, you will observe how Rational® Functional TesterHCL OneTest™ UI identifies the Graphical Editing Framework (GEF) palette objects and displays them as tree items in the script. You can use any of the GEF palette features to create the GEF objects. In this lesson, the zoom palette feature is being used.

About this task

Show Me

Lesson 3: Configure object recognition properties

In this lesson, you will see how to customize object recognition properties in the object library before you record scripts. Customizing the object recognition properties helps make the script resilient to changes in the application under test.

About this task

Show Me

Summary: Test GEF applications

This tutorial has shown you how to test applications based on the Graphical Editing Framework (GEF) using Rational® Functional TesterHCL OneTest™ UI.

Lessons learned

By completing this tutorial, you learned how to perform these tasks:

- Enabling the GEF applications.
- Testing the function of the GEF objects using Rational® Functional TesterHCL OneTest™ UI.
- Creating data, image and properties verification points on the GEF objects, given that Rational® Functional TesterHCL OneTest™ UI recognizes GEF edit parts and palettes.

- Identifying GEF test objects using the scroll logic
- Identifying GEF palette objects.
- Configuring object recognition properties.

Extend Rational® Functional TesterHCL OneTest™ UI capabilities using Proxy SDK

The movies in this tutorial show you how to extend the Rational® Functional TesterHCL OneTest™ UI proxies using the proxy SDK.

This tutorial shows you how to extend the Rational® Functional TesterHCL OneTest™ UI proxies using the proxy SDK to change the recording behavior for a button object. It uses Microsoft® Visual Studio Integrated Development Environment to create .NET classes and a sample .NET application that is a part of the .NET samples.

Learning objectives

This tutorial shows you how to extend a Rational® Functional TesterHCL OneTest™ UI proxy. Specifically, you will learn how to perform the following:

- Create and build a proxy project
- Create a customization file to map the proxy to an AUT control
- Deploy the proxy
- Verify the new proxy by recording a functional test script

Time required

40 minutes

Module 1: Extend Rational® Functional TesterHCL OneTest™ UI capabilities using Proxy SDK

The movies in this module show you how to extend the Functional Tester proxies using Proxy SDK.

Learning objectives

This tutorial shows you how to extend a Rational® Functional TesterHCL OneTest™ UI proxy. Specifically, you learn how to perform these tasks:

- Create and build a proxy project
- Create a customization file to map the proxy to the control of an application_under_test
- Deploy the proxy
- Verify the new proxy by recording a functional test script

Time required

This module requires approximately 25 minutes to finish.

Introduction: Extend Rational® Functional TesterHCL OneTest™ UI capabilities using Proxy SDK

This tutorial is designed to introduce you to Rational® Functional TesterHCL OneTest™ UI Proxy SDK.

Learning objectives

This tutorial is divided into five lessons. You will learn to extend the Rational® Functional TesterHCL OneTest™ UI proxies using the Proxy SDK.

- Record a functional test script to observe the existing recording behavior with the default proxy class used by a button.
- Create a proxy class and build a dll file
- Create a customization file to map the proxy control
- Deploy the binaries into the Rational® Functional TesterHCL OneTest™ UI customization folder
- Verify the new proxy by recording a functional test script

This tutorial is intended for the advanced users familiar with Rational® Functional TesterHCL OneTest™ UI framework and has knowledge of .NET programming.

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Lesson 1: Record a functional test script

In this lesson, you will record a functional test script to observe the recording behavior with the default proxy class used by a button.

About this task

Show Me

Lesson 2: Create and build a proxy project

In this lesson you will learn how to create and build a proxy project.

About this task

Show Me

Lesson 3: Create a customization file to map the proxy to an AUT control

In this lesson, you will learn how to create a customization file and map the proxy to an AUT control.

About this task

Show Me

Lesson 4: Deploy the proxy

In this lesson, you will learn how to deploy the .jar and .rftcust files to the Functional Tester customization folder.

About this task

Show Me

Lesson 5: Verify the new proxy by recording a functional test script

In this lesson, you will record a functional test script similar to lesson 1 and verify the new proxy. You will view the difference in the recording behavior when the new proxy class is used by the button object.

About this task

Show Me

Module 2: Develop proxies using Proxy SDK wizards

The movies in this tutorial show you how to develop Rational® Functional TesterHCL OneTest™ UI proxy with the Proxy SDK wizards. You can create proxy stubs using the Proxy SDK wizards.

Learning objectives

This tutorial shows you how to extend a Rational® Functional TesterHCL OneTest™ UI proxy. Specifically, you will learn how to perform these tasks:

- Create a proxy project
- Create a proxy class and build it
- Export the proxy package
- Import the proxy package
- Verify the new proxy by recording a functional test script

Time required

This module requires approximately 30 minutes to finish.

Introduction: Develop proxies with Proxy SDK wizards

This module demonstrates how you can use the Proxy SDK Wizards to create proxy classes, map controls to the proxy, create template proxy code, and deploy the proxy. This works only in case of domain extensions implemented in Java™. Using the export and import options available in the Proxy SDK Wizards, you can create a proxy package and automate the deployment and loading of the proxy in Rational® Functional TesterHCL OneTest™ UI respectively.

Learning objectives

This tutorial is divided into six lessons. In the tutorial you learn to extend Functional Tester capabilities by using the Proxy SDK wizard to test the custom control. In this tutorial you learn to perform these tasks:

- Record a functional test script
- Create a proxy project
- Create a proxy class and build it
- Export the proxy package
- Import the proxy package
- Verify the custom proxy

Time required

This tutorial requires approximately 30 minutes to finish. If you explore other concepts that are related to this tutorial, it might take longer to complete.

Skill level

Advanced

Audience

This tutorial is intended for advanced users who are familiar with Rational® Functional TesterHCL OneTest™ UI Proxy SDK and who have knowledge of Java programming.

Lesson 1: Record a functional test script

In this lesson, you watch the recording of a functional test script and observe the default proxy class name of the custom tree control. For this tutorial, you see how to use the sample interface that contains the TreeApp application for recording.

About this task

Show Me

Lesson 2: Create a proxy project

In this lesson, you see how to create a proxy project.

About this task

Show Me

Lesson 3: Create a proxy class

In this lesson, you see how to create a proxy class and build it.

About this task

Show Me

Lesson 4: Export the proxy package

In this lesson, you see how to export the proxy package to the folder that the user specified. You can use the export wizard to export proxy resources to a compressed (.zip) file.

About this task

Show Me

Lesson 5: Import the proxy package

In this lesson, you see how to import the items in a proxy package into Rational® Functional TesterHCL OneTest™ UI or any computer. The jar file and the .rftcust file is imported into the Rational® Functional TesterHCL OneTest™ UI customization folder, for example, `C:\Documents and Settings\All Users\Application Data\IBM\RFT`.

About this task

Show Me

Lesson 6: Verify the custom proxy

In this lesson, you see how to record a functional test script similar to lesson 1 and verify the new proxy. You will see the change in class name after deploying the new proxy.

About this task

Show Me

Summary: Extend Functional Tester capabilities using Proxy SDK

This tutorial has shown you how to create a proxy project, build and deploy the proxies and verify the new proxy by recording a functional test script.

Lessons learned

By completing this tutorial, in Module1, you learned how to:

- Create and build a proxy project
- Create a customization file to map the proxy to the control of an application_under_test
- Deploy the proxy
- Verify the new proxy by recording a functional test script

In Module 2, you learned how to:

- Create a proxy project
- Create a proxy class and build it
- Export the proxy package
- Import the proxy package
- Verify the new proxy by recording a functional test script

Additional resources

If you want to learn more about the topics covered in this tutorial, consult the following resources:

- Functional Tester Proxy SDK Help
- Functional Tester Proxy SDK Samples

Tutorials for testing in the Visual Studio IDE

Rational® Functional TesterHCL OneTest™ UI tutorials contain educational materials to help you learn the technologies and concepts behind products.

Create a functional test

This tutorial teaches you how to get started using Rational® Functional TesterHCL OneTest™ UI and walks you through the major use cases for testing and performing basic operations. This tutorial uses the sample application provided with Rational® Functional TesterHCL OneTest™ UI to perform all the tasks.

The Rational® Functional TesterHCL OneTest™ UI tutorial is divided into 10 lessons that must be completed in sequence for the tutorial to work properly.

Learning objectives

After completing this tutorial, you will be able to:

- Create a functional test project and record a script
- Work with verification points, object maps, and regular expressions
- Use a comparator to update a verification point
- Play back a script
- Perform regression tests

Note: Consider printing the tutorial before you begin and using the printed copy as you work through the lessons. You can print each lesson by right-clicking inside each topic and clicking **Print**.

Time required

This tutorial should take approximately 45 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Prerequisites

This is an introductory tutorial. You should be able to perform the tasks with little or no experience with Rational® Functional TesterHCL OneTest™ UI.

Lesson 1: Set up Rational® Functional TesterHCL OneTest™ UI

Rational® Functional TesterHCL OneTest™ UI provides a Java™ Runtime Environment (JRE) that is installed and enabled for testing Java™ applications. You use that JRE for the tutorial. When you want to test your own Java™

or HTML applications, you must run the enabler and configure your environments and applications. For more information on these set-up tasks, see Getting Started with Rational® Functional TesterHCL OneTest™ UI. For now you do not need to do anything to use the preconfigured JRE to continue.

Start Rational® Functional TesterHCL OneTest™ UI and then perform the following tasks before you record your first test script.

Setting logging options

Rational® Functional TesterHCL OneTest™ UI provides several logging options. We will use the HTML log.

1. To verify that this is the logging option you have set, click **Tools > Options**.
2. In the left pane of the Options dialog box, expand **Functional Test**, then **Playback**, and click **Logging**.
3. Select **html** as the**Log type**
4. Click **OK**.

This setting opens the HTML log automatically after you play back a script.

Create a Functional Test project

Before you start recording, you must create a Functional Test project.

1. In the product menu, click **File > New > Functional Test Project**.
2. Under **Name**, type `FTtutorial` (no spaces).
3. Under **Location**, type `C:\FTproject`. Rational® Functional TesterHCL OneTest™ UI creates this directory.
4. If the source control option is available, do not select **Add the project to Source Control**.
5. If the associate project option is available, do not select **Associate the Functional Test Project with current Project**.
6. Click **OK**.

The FTtutorial project is now visible in the Solution Explorer, which is the right pane in the Rational® Functional TesterHCL OneTest™ UI window.

Advance to Lesson 2: Record a script on page 148.

Lesson 2: Record a script

In this lesson, you will record a script using the Rational® Functional TesterHCL OneTest™ UI Recording Monitor.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

Begin recording

You are now ready to begin recording.

1. To start recording, click the **Record a Functional Test Script** button  in the product toolbar.
2. In the **Name** field, type `Classics` (Classics is the name of the application you will be using).

3. Do not select the **Add script to Source Control** option if it is available.
4. Click **Open**.
5. In the Select Script Assets dialog box, keep all defaults and click **Finish**.

   The Rational® Functional TesterHCL OneTest™ UI window automatically minimizes, and the Recording Monitor appears.

   Note: The Rational® Functional TesterHCL OneTest™ UI Recording Monitor is displayed every time you begin recording. You can minimize the monitor if you don't want it to be visible on the screen, and you can also resize it. You can also click the Display Toolbar Only button , which hides the monitor and shows only the toolbar. Leave the monitor displayed during this tutorial. For more information about the monitor and its toolbars, see The Rational® Functional TesterHCL OneTest™ UI Recording Monitor on page 1807.

6. Click the **Monitor Message Preferences** toolbar button . You can use these options to control the appearance of the text in the monitor any time.
7. Click **Cancel**.
8. Click the **Insert Script Support Commands** toolbar button . This opens the Script Support Functions dialog box on page 890, which allows you to call another script, insert a log entry, insert a timer, insert a sleep command (a delay), or insert a comment into your script.
9. Click **Close**.

Start the application

1. Click the **Start Application** toolbar button .
2. In the Start Application dialog box, use the arrow to select **ClassicsJavaA**, if necessary, and click **OK**.

The Rational® Functional TesterHCL OneTest™ UI Tutorial sample application, ClassicsCD, opens.

If the Recording Monitor is in front of the application, you can click the monitor's title bar and drag it to the lower right corner of the screen.

Record actions

You are going to record placing an order in this application

1. Click the **+** next to **Haydn** to expand the folder in the **Composers** tree.
2. In the list, click **Symphonies Nos. 94 & 98**.
3. Click the **Place Order** button.
4. In the Member Logon dialog box, keep the default settings of **Existing Customer** and **Trent Culpito**.
5. Do not click either of the password fields at this time. Click **OK**.
6. In the **card number** field, enter a credit card number. You must use the valid format of four sets of four digits here, for instance, `7777 7777 7777 7777`.

7. In the **expiration date** field, enter a valid format expiration date, `06/09`.

   Note: Date can be same or later than the system date.

8. Click **Place Order**.
9. Click **OK** in the order confirmation message box.

Advance to Lesson 3: Create verification points on page 150.

Lesson 3: Create verification points

In this lesson, you will record verification points to test objects. Verification points verify that a certain action has taken place, or verify the state of an object.

You can create a Properties verification point or six types of Data verification points. When you create a verification point, you capture information about an object in the application to establish baseline information for comparison during playback. You will record a Data verification point to capture the tree of composers.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

Create a data verification point

1. In the Recording Monitor, click the **Insert Verification Point or Action Command** button .
2. In the Select an Object page of the Verification Point and Action Wizard, clear the **After selecting an object advance to next page** option if it is selected.
3. Use the Object Finder to select the Composers tree in the application. Click the Object Finder and drag it over the tree. While holding down the mouse button, you will see that the entire tree is outlined with a red border and the object name is displayed (`javax.swing.JTree`) in a screen tip next to the red border. When you release the mouse button to make the selection, notice that the recognition properties for the object are listed in the grid at the bottom of the **Select an Object** page.
4. Click **Next**.
5. In the Select an Action page, make sure **Perform Data Verification Point** is selected and click **Next**.
6. On the **Insert Verification Point Data Command** page, in the **Data Value** field, select the **Tree Hierarchy** test. This test captures information about the entire tree hierarchy.
7. In the **Verification Point Name** field, type `Classics_tree` and click **Next**.
8. The **Verification Point Data** page displays the captured data in a grid in the right pane. If a check mark appears in the box beside an item, that item will be tested. By default, all items are selected. Leave them checked. If they are not selected, click the **Check All** button.
9. Click **Finish**.

Create an image verification point

You can insert an image verification point to confirm that the appropriate album is displayed for the selected CD.

1. In the Recording Monitor, click the **Insert Verification Point or Action Command** button (
2. In the Select an Object page of the Verification Point and Action Wizard, clear the **After selecting an object advance to next page** option if it is selected.
3. Use the Object Finder ( ) to select the Album image in the application. Click the **Object Finder** and drag it over the album image. While holding down the mouse button, you will see that the album image outlined with a red border and the object name is displayed (`javax.swing.JLabel`) in a screen tip next to the red border. When you release the mouse button to make the selection, notice that the recognition properties for the object are listed in the grid at the bottom of the Select an Object page.
4. Click **Next**.
5. In the Select an Action page, select**Perform Image Verification Point** and click **Next**.
6. In the Insert Image Verification Point Command page, type `Album_image` as the **Verification Point Name**.
7. Make sure that the option **Select full image** is selected and click **Next**.
8. The Verification Point Data page displays the captured image in the right pane. Click **Finish**.

Create a properties verification point

You can now insert a different verification point to confirm that the order is for the correct customer. A Properties verification point is used to capture the text in the confirmation screen.

1. In the ClassicsCD application, click **Order > View Existing Order Status**. Do not click either of the password fields at this time.
2. Click **OK**.

   You will test on the label **Order for Trent Culpito** in the View Existing Orders dialog box.

3. In the Recording Monitor, click the **Insert Verification Point or Action Command** button .
4. On the **Select an Object** page, select the advance to next page option this time (the check box beneath the Object Finder).
5. Drag the Object Finder over the label, **Order for Trent Culpito**, to select it. While holding down the mouse button, you'll see that the label is outlined with a red border and the object name is displayed (`javax.swing.JLabel`).

   After you select the object, the **Select an Action** page opens because you selected the advance to next page option.

6. Select **Perform a Properties Verification Point**, which is the second action from the top. Click **Next**.
7. On the **Insert Properties Verification Point Command** page, leave the **Include Children** field set to **None**.
8. Under **Verification Point Name**, accept the suggested default.
9. Leave the **Use standard properties** option checked, and click **Next**.

   On the **Verification Point Data** page, the test object properties and their values are displayed in a grid format. You can choose which properties to test in the **Property** column and can edit the property values in the **Value** column.

Note: By default, none of the properties are selected. To test object properties, you must select the properties you want to test by checking each of them. The properties you select are tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button above the grid. Use the **Uncheck All** button to clear all properties. When you use a Properties verification point, it is a good practice to test only the properties you are interested in. In this case, only the **text** property is of interest to determine whether the order is for the correct customer.

10. In the Property column select the **text**, **opaque**, and **visible** properties to test them during playback. You may have to click the check box twice for the selection to persist.
11. Click **Finish**.
12. In the ClassicsCD View Existing Orders dialog box, click **Close**.

Test the password fields

Now let us place another quick order to test the password fields that we did not test earlier.

1. Expand the **Schubert** folder in the composers tree.
2. Click **String Quartets Nos. 4 & 14**.
3. Click **Place Order**.
4. In the Member Logon dialog box, keep the default settings of **Existing Customer** and **Trent Culpito**.
5. This time, enter `xxxx` in the **Password** field.
6. Click the check box to select the **Remember Password** option.
7. Click **OK**.
8. Enter a valid format credit card number and expiration date, for instance, 7777 7777 7777 7777, expiration 06/09.
9. Click **Place Order**.
10. Click **OK** in the order confirmation message box.
11. Close the ClassicsCD application by clicking the **x** button.
12. Click the **Stop Recording** button on the **Recording** toolbar.

    When you stop recording, Rational® Functional TesterHCL OneTest™ UI closes the monitor, and writes your script and object map to your project directory. The Rational® Functional TesterHCL OneTest™ UI window is restored and the script displays in the main window.

Advance to Lesson 4: Play back the script on page 152.

Lesson 4: Play back the script

In this lesson, you will play back the script and look at some parts of the Rational® Functional TesterHCL OneTest™ UI interface. Because the script you just recorded is the active script, that script will play back when you click the playback button.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

Because the script you just recorded is the active script, that script will play back when you click the playback button.

1. To play back the script, click the **Run Functional Test Script** button on the product toolbar, or click **Script > Run**.
2. In the Select Log dialog box, keep the default log name of **Classics** and click **Finish**.

   The Rational® Functional TesterHCL OneTest™ UI window minimizes, and the Playback Monitor on page 1443 opens at the top right of your screen. As the script plays back, messages display in the Playback Monitor. Rational® Functional TesterHCL OneTest™ UI plays back all of your recorded actions, such as the application starting, the actions you performed on the application, and the verification points.

   When playback finishes, the HTML log displays the results of the run in a separate window. If the log is minimized, restore it. All events listed in the log should include **Pass** in the event headings in green. Notice that the two verification points you recorded are listed.
3. Close the log window.

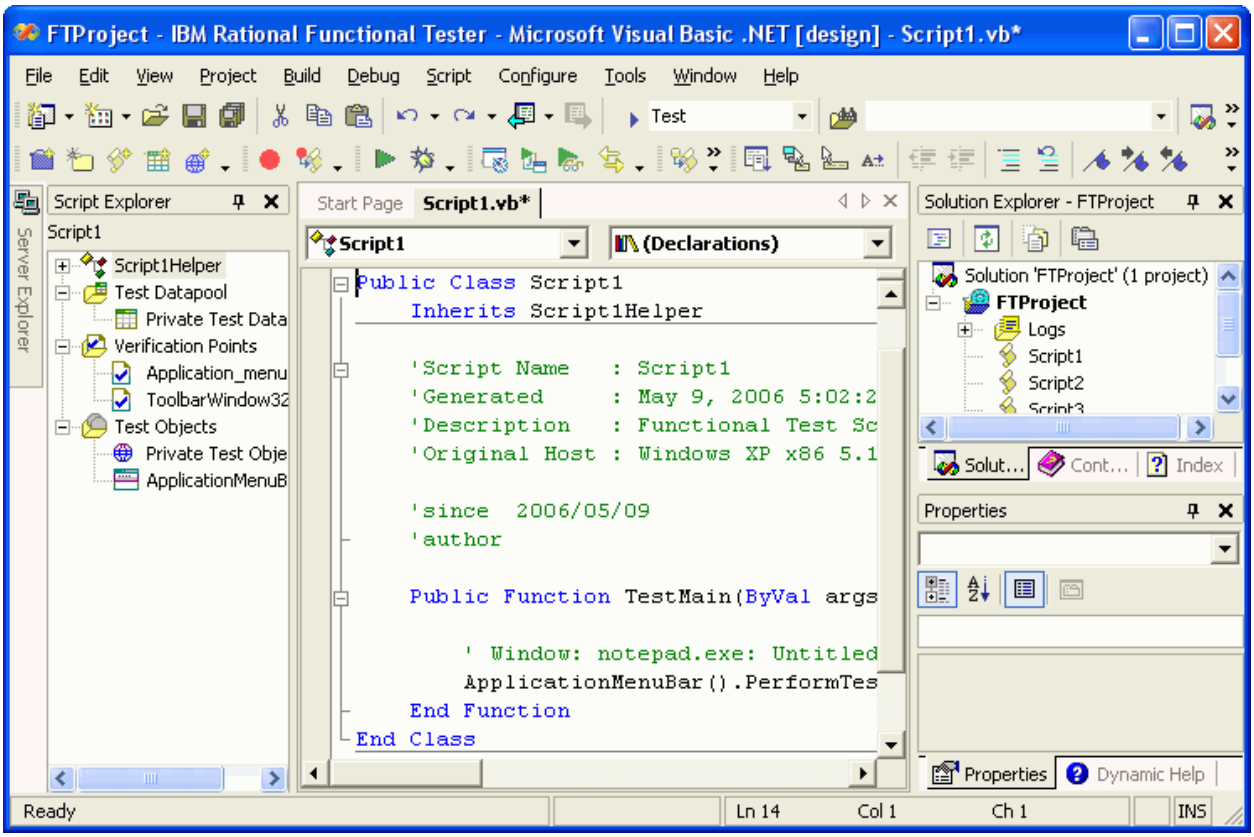
   Now that you have successfully recorded a script and played it back, let's look at the Functional Test perspective in more detail.
4. If the Rational® Functional TesterHCL OneTest™ UI window is minimized, restore it.

When you have multiple scripts, Rational® Functional TesterHCL OneTest™ UI displays all open scripts in a project in the Editor (the script window). Each script has a tab in the banner of the window, and the tab for the active script shows the script name in bold. Any actions you perform on a script, such as playing it back, are performed on the active script.

Throughout the script, notice the information about the script shown at the top in light blue and prefixed by asterisks. This information comes from the script template, which you can modify. For more information, see Customizing a Script Template on page 1166.

Notice also that Rational® Functional TesterHCL OneTest™ UI adds a short comment to the script in green characters to identify the object that the following lines refer to. This information makes it easier to navigate the script. Strings passed as arguments to methods during recording, including user inputs, are bright blue.

To the right of the Editor (the script window) is the Solution Explorer, which lists any Functional Test projects to which you are currently connected. All scripts within each project are listed below the project name. This Solution Explorer provides another way to navigate to a different script. When you double-click a script in the Solution Explorer, it opens in the script window and becomes the active script. For more information, see the Solution Explorer.



To the left of the Editor is the Script Explorer, which lists the verification points and object map of the active script. From the Script Explorer you can start the Verification Point Editor to display and edit verification points, and start the object map editor to display and edit object maps. For more information, see Script Explorer on page 1813.

For information about the other parts of the Rational® Functional TesterHCL OneTest™ UI window, see About the Visual Studio .NET Window on page 59.

Advance to Lesson 5: View verification points and object maps on page 154.

Lesson 5: View verification points and object maps

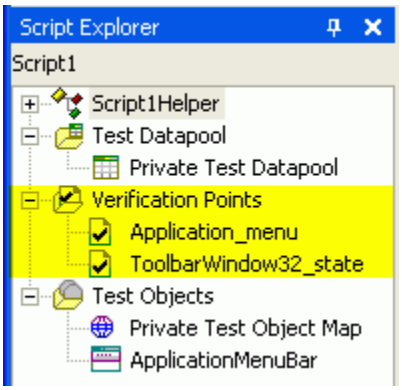
In this lesson, you will learn how to view and modify the properties of verification points and object maps.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

View verification points

You can examine and modify the data inside a verification point.

1. In Rational® Functional TesterHCL OneTest™ UI, verify that your script, Classics.vb, is still the active script in the Editor.
2. The two verification points you recorded should be listed in the Script Explorer to the left of the script. If necessary, click the plus sign (+) next to **Verification Points** to expand the folder.



3. Double-click **Classics_tree**.

   This is the first verification point you recorded, on the list of composers. The Verification Point Editor starts; you can update verification point data for future playbacks.

   The Data verification point has six possible display types. This verification point is a Data (tree) type. The object type is a tree, in this case, a javax.swing.JTree. To edit the data in the tree, double-click any of the sub-items in the tree to open a small edit box, where you can make changes. Use the check boxes beside each item to indicate whether you want this item to be tested in future playbacks. Checked items get tested. To learn more about using the Verification Point Editor, see the Verification Point Editor on page 917.

4. Close the Verification Point Editor.

View object maps

You can also examine and modify the data inside the object map.

1. In the Script Explorer, expand the **Test Objects** folder, if necessary.

   The first item, Private Test Object Map, is the script's object map. The individual objects listed beneath Private Test Object Map are references to objects that were acted on during recording.

2. Double-click **Private Test Object Map** to open it.

   When you record a script, Rational® Functional TesterHCL OneTest™ UI creates an object map for the application under test. Each script is associated with an object map file. The map file can be private -- associated with only one script -- or shared among many scripts. When you recorded the script, Rational® Functional TesterHCL OneTest™ UI used the default setting (private map) on the second tab of the Record a New Functional Test Script dialog box. The object map contains properties for each object, and you can easily update the information in one central location. Then, any scripts that reference that object also share the updated information.

   In your object map, the top-level object of a frame, lists all the objects the frame includes beneath the frame object.

3. Expand the top-level object "`Java: Frame: logFrame1: javax.swing.JFrame`."

   The frame object includes the logon dialog box. The radio buttons, password fields, and action button are listed beneath the frame object.

4. Click one of the objects.

   Notice that the recognition properties appear in the grid below the object tree.

   The object map also provides a quick way to add object references to a script. In the object map menu, you can click **Test Object > Insert Object(s)** to add objects. For more details, see Adding Objects to an Object Map.

   You can also perform other operations from the object map, such as changing the weight of a recognition property and editing recognition properties and values. We'll do several advanced procedures using the object map later on. For information about the parts of the object map and for links to procedures, see Test Object Map on page 1838.

5. In the object map menu, click **Preferences > Clear State On Close**.

   The **Clear State On Close** command is a toggle menu and should be selected by default, so you will be clearing it. If it were left on, when you close the map all objects would be accepted. We want to do that in a later step when we return to the object map to make changes.

6. Close the object map. Do not save any changes you may have made.

Advance to Lesson 6: Perform regression tests on page 156.

Lesson 6: Perform regression tests

In this lesson, you will execute your script on a different build. When you have a new build of the application, you can run the automated test you recorded by playing back your script on the new build. To execute your script on the new build, you must change the name of the application in your script. (You would not need to do this on a development project; do it here to simulate getting a new build of the application.)

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

1. In the script window, verify that your script (Classics.vb) is the active script.

   At the top of the script, beneath the template information, is the start application command:

   ```
   StartApp("ClassicsJavaA")
   ```

2. Change the "A" to "B".

   VB.Net code is case-sensitive, so be sure to use an uppercase B. You do not need to save or compile the script for the change to take effect. It is done automatically when you run the script.

3. Click the **Run Functional Test Script** toolbar button  to play back the script.

4. In the Select Log dialog box, select **Classics** if necessary and click **Finish**. You will be prompted to overwrite the log.
5. Click **Yes**.

   The script begins to play back quickly, but slows near the end on the Member Logon dialog box. That is because Build B of the application has different text in the field beside the check box. Rational® Functional TesterHCL OneTest™ UI is looking for an object that matches the recognition properties recorded in Build A. We will show how to fix this problem later on.
6. When the log opens after playback, look at the messages. You should see two failures and one warning in the log. (Keep the log open in preparation for lesson 7.)

   The properties verification point (OrderFor TentCulpito_standa) and the image verification point (Album_image) failed because of a change in the application. Next, we'll see how to update the verification point baseline to fix this. An object recognition warning was generated for the password check box field. We'll also show how to fix that in the object map using a regular expression in a later section of the tutorial.

   Did you notice that the main screen of ClassicsB looks different from ClassicsA, but that did not cause the script to fail? The same objects are present but in a different location in the two applications. This did not cause a failure because Rational® Functional TesterHCL OneTest™ UI uses robust recognition methods to locate the objects. For example, it does not rely on superficial properties such as screen coordinates to find objects. Instead, it uses internal recognition properties. This method allows for flexibility in the user interface design, without requiring that you alter or re-record your scripts.

Advance to Lesson 7: Use the Comparator to update a verification point on page 157.

Lesson 7: Use the Comparator to update a verification point

You can use the Verification Point Comparator to compare verification point data after you play back a script. Verification points provide a baseline of the properties or data of an object. If the verification point fails on a subsequent build of an application, you have found a defect or an intentional change to the application. If the change is intentional, you can update the information in the verification point so that the test continues to be valid for future builds.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

At the end of lesson 6, you left the log open. If you closed the log, reopen it by double-clicking on the log name in the Solution Explorer.

1. In the log, click the View Results link at the end of the failed image verification point entry. The event heading is "Verification Point (Album_image)."

   The Rational® Functional TesterHCL OneTest™ UI Verification Point Comparator displays your verification point data. Notice that the Comparator banner includes the name of your verification point.

Problems with the comparator?: If the comparator does not open or you get an error message, you need to enable the Java™ plug-in of your browser. For the instructions to do that, see the topic called "Enabling the Java™ Plug-in of a Browser" in the "Before You Record" section of the Rational® Functional TesterHCL OneTest™ UI Help.

When a verification point fails, the Comparator shows the expected and the actual values to help you analyze the differences. You can then load the baseline file and edit it or update it with the values from the actual file. Failures are displayed in red.

When you created the verification point on ClassicsA, the captured album image is based on the object `javax.swing.JLabel`. When you played back the script on ClassicsB, since the height and the width of the object `javax.swing.JLabel` is different, the image verification point failed. So you must update the baseline file to change the object to match ClassicsB.

2. Click the **Load Baseline to Edit** button () on the Comparator toolbar.
3. Click the **Replace Baseline with actual value** button () on the Comparator toolbar.The actual image is loaded as the baseline image.
4. Close the Comparator.
5. In the log, click the **View Results** link at the end of the failed verification point entry. The event heading is "Verification Point (OrderforTrentCulpito_standard)."

   The Rational® Functional TesterHCL OneTest™ UI Verification Point Comparator displays your verification point data. Notice that the Comparator banner includes the name of your verification point. (Note: if you get an error message and the Comparator does not open, click here on page 800.)

   When a verification point fails, the Comparator shows the expected and the actual values, to help you analyze the differences. You can then load the baseline file and edit it or update it with the values from the actual file.

   Failures are displayed in red.

6. Scroll to the **text** property.

   When you created the verification point on ClassicsA, the banner title was "Order for Trent Culpito." When you played back the script on ClassicsB, the banner title was "Orders for Trent Culpito." "Orders" is correct, because a customer may have multiple orders in the Orders dialog box. So you must update the baseline file to change the text to match ClassicsB.

   You can edit only the baseline file.

7. Click the **Load Baseline to Edit** toolbar button  on the Comparator toolbar.

   Notice that the left **Value** column displays the **Baseline Value** now.

8. Instead of scrolling to the **text** property, you can click the **Jump to First Difference** button  above the **Property** column.

The four navigation buttons help you locate the differences between the baseline and actual files.

You can update the baseline file in two ways. You can edit that cell of the grid, adding the letter *s* to the word "Order," or you can use the **Replace Baseline** command. Replacing the baseline replaces all values from the baseline file with the values from the actual file. In general, if you need to edit only one or a small number of values, you should edit the individual values.

9. This test has only one difference to update, so click the **Replace Baseline with actual value** button  on the Comparator toolbar.

   Both values in the **text** property now match and the property no longer appears in red.

   For more information about using the Comparator, see the Verification Point Comparator on page 923 or Comparing and Updating Verification Point Data Using the Comparator on page 933.

10. Close the Comparator.

    Now we will play back the script again to confirm the verification point passes, given the updated the baseline value for the failure.

11. Close the log first.

12. Click the **Run Functional Test Script** button.

13. Select the **Classics** log if necessary and click **Finish**.

14. Click **Yes** if prompted to overwrite the log.

    Rational® Functional TesterHCL OneTest™ UI pauses on the Member Logon dialog box, because you did not fix that recognition problem yet. At the end of playback, Rational® Functional TesterHCL OneTest™ UI displays the log. The verification point now passes! See how easy it is to use the Comparator to update object data and properties to account for changes in the application under test.

15. Leave the log open.

Advance to Lesson 8: Update the object map on page 159.

Lesson 8: Update the object map

In this lesson, you will fix the object recognition warning by using the object map. You will also use a regular expression for more flexible object recognition.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

When you see a recognition failure or warning, look at the log message. At the end of Exercise 7, you kept the log open. If it is not open, open it by double-clicking it in the Solution Explorer.

One warning remains in the log. The event heading is **Object Recognition is weak (above the warning threshold)**.

1. Look at the **ObjectLookedFor** and **objectFound** fields in the warning section near the bottom of the log.

   In ClassicsA, the name of the password field is **Remember Password**. In ClassicsB it is **Remember The Password**. When you played back the script on ClassicsB, the object recognition did not match exactly because of this difference.

2. Look at the **Line Number** field in the log. Note the number and close the log to return to Rational® Functional TesterHCL OneTest™ UI.
3. Click anywhere in the script window, and then click **Edit > Go to**.
4. Type the line number from the log failure message, and click **OK**.

   The cursor moves to the left margin of that line number.

   Note: You can also find the line number by looking at the indicator in the bottom of the Rational® Functional TesterHCL OneTest™ UI window. The line number and cursor position within the line is shown there. For example, "Ln 66 Col 9" refers to the position on line 66 that is 9 characters to the right of the left margin.

   The line in your script should be:

   ```
   RememberPassword().ClickToState(SELECTED)
   ```

   This line represents your click on the password check box. This line in the script shows which object is failing. Now you can look for that object in the object map.

5. To find the object, return to the list of Test Objects in the Script Explorer (left pane).

   You should see `RememberPassword` listed under the **Test Objects** folder.

View the object recognition properties in the object map

1. Double-click the **rememberPassword** object to open it in the object map.
2. Click **Test Object > Accept All** on the object map menu. If the command is grayed out, don't do anything.

   Notice that all the objects change to black text. The text is blue (to indicate new objects) until you accept the objects in a map. You should accept the objects the first time you look at a newly created object map.

3. If the password check box object is not selected in the map, select it. (It is the object called `Java: checkBox: checkRemeber: javax.swing.JCheckBox`.)
4. Look at the recognition properties listed in the **Recognition** tab at the bottom of the object map.

   You can see that this is the object from ClassicsA, because it says `Remember Password` in the **accessibleContext.accessibleName** property. This is the old object. However, when you played back the script on ClassicsB, the text for that object changed, so Rational® Functional TesterHCL OneTest™ UI recognizes it as a new object. You want to use the new object properties in this case, so you must add it to the map.

Add the new object to the map

To add the new object to the map, open ClassicsB and the Member Logon dialog box.

1. Click **Applications > Run** in the object map menu.
2. Use the arrow to select **ClassicsJavaB**. (Be sure to pick B.)
3. Click **OK**.
4. In ClassicsCD, select any CD and click **Place Order**.

   The Member Logon dialog box opens.
5. Move the object map lower on your screen, if necessary, to see all of it. In the object map menu, click **Test Object > Insert Object(s)**.

   This is the same as the Object Finder tool in the **Select an Object** page of the Verification Point Wizard.
6. Clear the advance to next page option if it is selected.
7. Use the Object Finder tool to select the **Remember the Password** check box in the Member Logon dialog box.

   After you select the check box, you'll see that the **accessibleContext.accessibleName** property is now "Remember The Password." Stretch the borders of the object map, if necessary, to see the properties.
8. On the **Select an Object** page, click **Next**.
9. Don't change anything on the **Select Object Options** page, and click **Finish**.

   The new check box object is now shown in the object map.
10. Click another object and notice that the new item is listed in blue and the word "New" appears at the beginning of the line.

    Now both the old and the new objects are listed in the map. You want to unify the two objects and take the properties from each that you want for the new object.

Unify the objects

1. To unify the objects, click the old object (the original check box labeled "CheckBox: checkRemember"), and drag it onto the new object in the list. Position the tip of the cursor arrow over the new object before you release the mouse button.

   The Unify Test Objects wizard appears.
2. Widen the Unify wizard if necessary to see more of the information in the lower sections.

   In the lower left section, the original object's properties appear. It should be labeled "Source: RememberPassword." That is what the text was on the check box in ClassicsA. In the lower right section, it should be labeled "Target: RememberThePassword." That is what the text is on the check box in ClassicsB. Because you dragged the old object to the new object, the new object's recognition properties are filled in at the top. In general, Rational® Functional TesterHCL OneTest™ UI puts the new properties at the top if they are the preferred properties. However, some old administrative properties may be preferred. For example,

Rational® Functional TesterHCL OneTest™ UI retains regular expressions in the old property set. To use a property from the old object, double-click that property in the grid of the old object and it will be copied up into the unified object. In this case, we want to use all the properties of the new object, which are already filled in.

3. Click **Next**.

   All scripts that are affected by this change in the object map are listed. Only one script, Classics, is affected.

4. Click **Finish**.
5. In the object map, click **File: Save** to save the changes you made, and close the object map.

Play back the script again

Now we'll play back the script again on ClassicsB to confirm that it passes.

1. First close both dialog boxes of ClassicsCD.
2. In Rational® Functional TesterHCL OneTest™ UI, click **Run Functional Test Script**.
3. Select the **Classics** log and click **Finish**.

   The script now passes with no warnings! Notice that the playback no longer pauses on the check box object because the recognition properties now match.

   This object unification feature is an easy way to update scripts when recognition properties of an object intentionally change. One of the major advantages of this feature is that if your object map is being used by many scripts, you could update them all when you make the change in the wizard. Instead of manually editing multiple scripts, you can make the change once in the map and it propagates automatically to all scripts that use it. This feature can save you time.

   Note: There is also an easier way to update the recognition properties of a test object should they change. Instead of using the Unify Wizard as described in this step, from the Object Map you can simply select the test object whose recognition properties you want to update. Right-click on the test object as it is displayed in the Object Map tree and select **Update Recognition Properties** from the pop-up menu. You will need to have the test application running when this action is performed so that Rational® Functional TesterHCL OneTest™ UI can get the updated recognition properties. You would only use this update method if you do not want to use any properties of the old object.

4. Close the log.

Advance to Lesson 9: Change the Recognition Preferences on page 162.

Lesson 9: Change the recognition preferences

In the previous exercise you saw how you can update the recognition properties of an object when they change. Another factor you can change is the recognition weights that Rational® Functional TesterHCL OneTest™ UI uses during playback. You use the recognition preferences to set this. The label object that you tested with the second verification point can demonstrate how this works.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

1. On the product menu, click **Tools > Options**.
2. Click **Functional Test > Playback > ScriptAssure**.
3. Click the **Advanced** button.

   Notice that one of the default settings is **Warn if accepted score is greater than: 10000**. A score of 10000 indicates that one important property can be wrong. Let's lower the score to 5000 and see what happens.
4. Click the **Use Default** check box beside this field.
5. Then type `4000` in the field. Click **OK**.
6. Play back the script on ClassicsB again.

   The log now contains a warning for the label object. The reason given in the **objectFound** field, is that the recognition score is 10000. The discrepancy was caused by changing the word "Order" to "Orders" in the label.
7. Close the log.
8. Restore the default value for the recognition score.
   a. Click **Tools > Options**.
   b. Click **Functional Test > Playback > ScriptAssure**.
   c. Click the **Advanced** button.
   d. Click the **Use Default** check box beside the **Warn if accepted score...** field.

      This will change the 4000 back to 10000.
   e. Click **OK**.
   f. Play back the script again.

      Now the warning is gone, and everything passes.
   g. Close the log.

This lesson showed how you can tweak the recognition score in order to achieve the sensitivity that you want for object recognition. For more information, see Using ScriptAssure™ on page 1441.

Advance to Lesson 10: Use regular expressions on page 164.

Lesson 10: Use regular expressions

The last thing you will do using the object map is convert a property value to a regular expression. In this case, the regular expression provides more flexibility in the object recognition.

To start at the beginning of the tutorial, see the Rational® Functional TesterHCL OneTest™ UI Tutorial on page 147.

We just saw how the script passes completely on ClassicsB now. That was our goal because the changes made to the application in ClassicsB are correct. So the script is now in the state we want it to be in going forward. Now when you play it back against ClassicsA, the script fails because of the changes made earlier. You might want to allow more than one variant of an object to pass. You may have a dynamic object or have several versions of your application with slightly different versions of an object, in which both are correct. You can use a regular expression to allow more than one version of a property value, such as text, to accommodate this scenario.

Open the object map and unify the objects

1. To play back against ClassicsA, edit the StartApp command at the top of the script and change the B to an A.
2. Click **Run Functional Test Script** on the Rational® Functional TesterHCL OneTest™ UI toolbar.

   During playback, Rational® Functional TesterHCL OneTest™ UI pauses briefly on the check box object, but eventually it finishes. The script now gives a warning. Notice in the log that the warning is for the same object, the password check box.
3. Close the log and then open the object map from the password check box object as you did in Lesson 8, by double-clicking **rememberPassword** in the Script Explorer.
4. In the object map, open the application by clicking **Applications > Run**.
5. Select **ClassicsJavaA** and then click **OK**.
6. Pick any CD and click **Place Order** in ClassicsCD to open the Member Logon dialog box.
7. Add the new object into the map by clicking **Test Object > Insert Object(s)**.
8. Use the Object Finder to select the password check box in the Member Logon dialog box in the application.
9. Click **Next**, and then **Finish**.
10. In the top pane of the object map, drag the old check box object to the new check box object to unify the objects.
11. Widen the Unify Test Objects wizard by dragging one of the sides outward to make the fields longer, if necessary.

    You will use two different regular expressions: one on the **name** property and one on the **text** property.

    The unified object is shown in the **Unified Test Object Properties** grid (top pane); the **name** property has a value of "checkRemember."

Convert a property value to a regular expression

1. In the top pane, right-click the **checkRemember** value and click **Convert Value to Regular Expression**.

   Rational® Functional TesterHCL OneTest™ UI designates the value as a regular expression by the "xy" icon in front of the value text.

2. Double-click the **name** value again so that you can edit the field.
3. Delete the word `check`, and edit the remainder to read:

   [rR]emember

4. Click outside of that cell.

   This pattern allows the word "remember" with either an uppercase "R" or lowercase "r" to pass. This is important because the comparisons are case-sensitive, and only an exact match will pass.

   The value of the **text** property is "Remember Password."

5. Right-click the **Remember Password** value and select **Convert Value to Regular Expression** to convert it.
6. Then double-click it and edit it to read:

   Remember.*Password

   You are removing the space and adding the period (.) and asterisk (*) characters.

7. Click another cell.

   The "**.**" allows any character to appear in that position. In one version of the application, there is a space between the two words in this property, and in the other there is no space. This pattern covers both cases.

8. Click **Next**, and then **Finish** in the wizard.
9. Click **File > Save** in the object map to save the changes, and close the object map.
10. Close ClassicsCD.

    Now here comes the fun part!

11. Play back the script again on ClassicsA. If necessary, click anywhere inside the Rational® Functional TesterHCL OneTest™ UI window to activate the product toolbar, then click the **Run Functional Test Script** button. The image verification point and the properties verification point fails.The image verification point fails because the height and the weight of the object`javax.swing.JLabel` is different. The properties verification point is expected to fail because the text `Orders for Trent Culpito` was never changed to a regular expression.

    The object recognition warning on ClassicsA no longer appears in the log.

12. Close the log.
13. Change the StartApp command to play back ClassicsB, and run the script.

The object recognition also passes on ClassicsB! Regular expressions offer more flexible recognition for an object that has different properties in different versions of an application, and both are recognized during playback.

For more information about the object map, see Test Object Map on page 1838.

For more information about regular expressions, see Replacing an Exact-Match Property With a Pattern.

Tutorial summary

This Rational® Functional TesterHCL OneTest™ UI tutorial has shown you how to set up Rational® Functional TesterHCL OneTest™ UI for testing, recording and playing back scripts, creating verification points and using the Verification Point Comparator to update object properties or data, and several ways to use the object map to your advantage. There are many other features and use cases of Rational® Functional TesterHCL OneTest™ UI, such as the ClearCase® integration and cross-platform playback. Look through the table of contents in the Rational® Functional TesterHCL OneTest™ UI User Guide for more information about using Rational® Functional TesterHCL OneTest™ UI.

Create a data-driven functional test

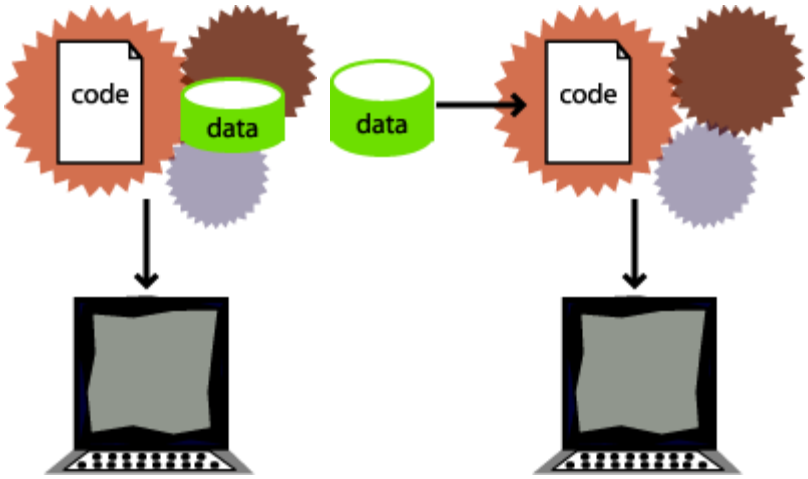
In this tutorial, you will learn how to data-drive a test using a variety of realistic data to test the application with Rational® Functional TesterHCL OneTest™ UI data-driver wizard.

Data-driven testing puts a layer of abstraction between the data and the test script, eliminating literal values in the test script. Because data is separated from the test script, you can:

- Modify test data without affecting the test script
- Add new test cases by modifying the data, not the test script
- Share the test data with many test scripts

A dataset is a collection of related data records. A dataset supplies data values to the variables in a test script during test script playback. Data-driven testing uses data from an external file, a dataset, as input to a test.

The diagram on the left shows a test script, which uses data with hard-coded, literal references in the test script. The diagram on the right shows a data-driven test script that uses data from an external file, a dataset.



Hard-coded test script **Data-driven test script**

About the sample application -- Classics CD

You will use the ClassicsCD sample application to create a project and record a test script to verify that the ClassicsCD sample application correctly totals an order placed in the application. You will also create a verification point with a dataset reference to check that the total amount of the order is correct in the Classics CD application.

Learning objectives

This tutorial is divided into several lessons that must be completed in sequence for the tutorial to work properly.

While completing the lessons, you will:

- Create a project and record a test script on page 168
- Data-drive a test on page 169
- Add descriptive headings to the data on page 169
- Create a verification point with a dataset reference on page 171
- Add data to the dataset on page 172
- Play back the test on page 174

Note: Consider printing the tutorial before you begin and using the printed copy as you work through the lessons. You can print each lesson by right-clicking inside each topic and then clicking **Print**.

Time required

This tutorial should take approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Advance to Lesson 1: Create a project and record a test script. on page 168

Lesson 1: Create a project and record a test script

In this lesson, you will use the Classics CD sample application to create a new project and start recording a data-driven test to verify that the sample application correctly totals the amount of music CDs purchased.

A project is a collection of test assets such as test scripts, object maps, verification points, and datasets, that can facilitate the testing of one or more software components. You must create a Functional Test project before you can record a test.

Create a project

Create a project to store the test assets you need to test the Classics CD sample application.

1. Start Rational® Functional TesterHCL OneTest™ UI.
2. Click **File > New > Functional Test Project**.
3. Type `DataDriveTutorial` for the name of the new project.
4. Click **Finish**.

Start recording

Start recording a test script to verify that when a customer orders a music CD, the total amount charged to the credit card is the correct amount listed in the application.

1. On the Functional Test toolbar, click **Record a Functional Test Script**( ).
2. Type `OrderTotal` for the name of the test script..
3. Click **Next**.

   The Select Script Assets page opens.

   When you create a test script, Rational® Functional TesterHCL OneTest™ UI creates a test dataset and other test assets. Use the defaults on this page: **Private Test dataset** and **Sequential**. A private test dataset is associated with only one script and is not shared by any other scripts. When you use the sequential order, the test script accesses records in the dataset in the order that they appear in the dataset.

4. Click **Finish**.

   The Rational® Functional TesterHCL OneTest™ UI window minimizes and the Recording Monitor opens.

Start the ClassicsCD application

Start the ClassicsCD application and navigate through the application to the dialog box that you will data-drive.

1. On the Recording toolbar, click **Start Application**( ) to start an application.
2. If necessary, click the **Application Name** arrow, to see the options, and then select **ClassicsJavaA - java**.
3. Click **OK**.

ClassicsJavaA is build 1 of the sample application, ClassicsCD, which comes with Rational® Functional TesterHCL OneTest™ UI.

The sample application starts.

4. In the **ClassicsCD** application, under **Composers**, double-click **Schubert** to open the CDs for sale by that composer, and then click **String Quartets Nos. 4 & 14**.
5. Click **Place Order**.
6. Click **OK** to close the Member Logon dialog box.

   The Place an Order dialog box opens.

7. In the ClassicsCD application, type `1234567890` in the **Card Number** box and `09/09` in the **Expiration Date** box.

Advance to Lesson 2: Data-drive a test on page 169.

Lesson 2: Data-drive a test

In this lesson, you will use the data-driver to populate a dataset with data from the sample application.

1. On the **Recording** toolbar, click **Insert Data Driven Commands** ().

   The test script recording pauses and the Insert Data Driven Actions page opens.

2. From the Insert Data Driven Actions page, use the mouse to drag the hand, the Object Finder (), to the title bar, **Place an Order**, on the ClassicsCD application.

   Rational® Functional TesterHCL OneTest™ UI outlines the entire Place an Order dialog box with a red border.

3. Release the mouse button.

   The Data Drive Actions page opens. In the Data Drive Actions page, under the **DataDriven Commands** table, information appears about the objects you selected.

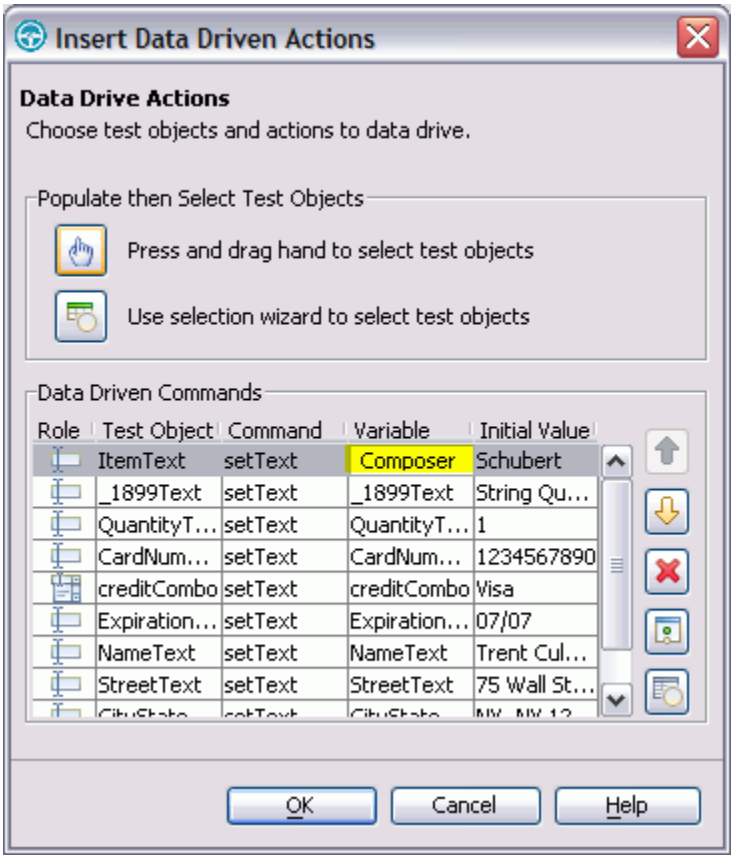
   You can hover over a row in this table to view the line of code that Rational® Functional TesterHCL OneTest™ UI inserts into the test script to data-drive the test script.

Advance to Lesson 3: Add descriptive headings to the data. on page 169

Lesson 3: Add descriptive headings to the data

In this lesson, you will add descriptive headings to the dataset you created in the previous lesson. Descriptive headings will make it easier to add data to the dataset.

1. In the **Data Driven Commands** table, under the **Variable** header, replace **ItemText** with `Composer`.



2. Repeat sequentially, replacing each cell in the **Variable** column with a descriptive name for each heading in the **Variable** field. Use the text in the following variables list as descriptive names.

Note: Do not use spaces in **Variable** names. Typically, you would look at the application to determine the appropriate headings for each row, but we have done that for you in the following table:

Variable
Composer
Item
Quantity
CardNo
CardType
ExpiryDate
Name
Street

Variable
CityStateZip
Phone

Rational® Functional TesterHCL OneTest™ UI automatically updates the test script as you change each of the Variable names.

3. Click **OK**.

   The Insert Data Driven Actions page closes.

   Now the dataset has descriptive headings that make it easier to add more data. You will add more data to the dataset after you finish recording the test script.

Advance to Lesson 4: Create a verification point with a dataset reference. on page 171.

Lesson 4: Create a verification point with a dataset reference

In this lesson, you will create a verification point with a dataset reference to check that the total amount due for the order is correct in the Classics CD application.

A verification point captures object information and literal values from the application-under-test and stores it as the baseline for comparison during playback. When you play back the script, a verification point captures the object information again to compare it to the baseline and see if any changes have occurred, either intentionally or unintentionally. Comparing the actual object information in a script to the baseline is useful for identifying potential defects.

You will use a dataset reference instead of a literal value for the value that you are testing in the verification point. Using datasets with verification points gives you more flexibility to test realistic data with your test scripts.

Create a verification point with a dataset reference

1. On the **Recording** toolbar, click **Insert Verification Point or Action Command** ().

   The Verification Point and Action Wizard opens.

2. From the Verification Point and Action Wizard, use the mouse to drag the **Object Finder** () to $19.99, next to Total, in the Classics CD application.

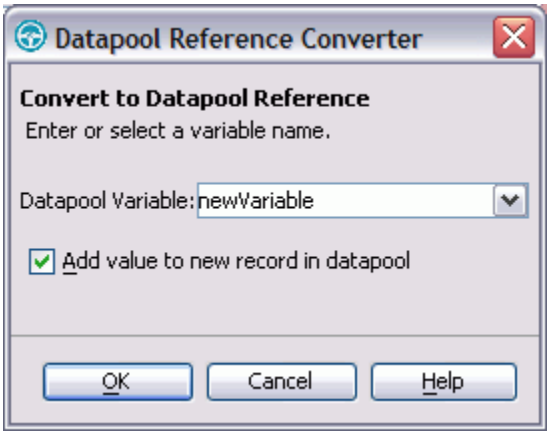
   Rational® Functional TesterHCL OneTest™ UI outlines $19.99 with a red border.

3. Release the mouse button.
4. If the Select an Action page is not displayed, click **Next**.
5. On the Select an Action page, if necessary, click **Perform Data Verification Point** to test that the total amount equals the expected amount.
6. Click **Next**.

The Insert Verification Point Data Command page is displayed.

7. Click **Next**.
8. On the Verification Point Data page toolbar, click **Convert Value to dataset Reference** ( ) to use a dataset instead of a literal value in a verification point. (If you cannot see the **Convert Value to dataset Reference** button on the toolbar, make the page larger by dragging a corner of the page)

   The dataset Reference Converter dialog box opens.



9. In the dataset Variable box, type `Total` to replace the **newVariable** for the heading in the dataset.
10. If necessary, select the **Add value to new record in dataset** check box to add the **Total** to the existing dataset record you created in the previous exercise.
11. Click **OK**.
12. Click **Finish**.

Place the order and close the ClassicsCD application

1. In the **ClassicsCD** application, click **Place Order** to place the order, and then click **OK** to close the message confirming your order.
2. Click **X** in the upper right corner of the **Classics CD** application to close the application.

Stop recording

On the **Recording** toolbar, click **Stop Recording** ( ) to write all recorded information to the test script.

The test script appears in the editor window.

Advance to Lesson 5: Add data to the dataset. on page 172

Lesson 5: Add data to the dataset

In this lesson, you will add data to the dataset to test that the ClassicsCD sample application correctly totals each order placed in the application.

1. In the Script Explorer, double-click **Test dataset** and then double-click **Private Test dataset**. Under the test script editor, double-click the **Test dataset** tab to expand the dataset editor so that you can work.

   The dataset editor opens and should look similar to the following table:

	Composer	Item	Quantity	Card#	CardType	ExpDate	Name	Street	CityStZip	Phone	Total
0	Schubert	String Quartets Nos. 4 & 14	1	1234567890	Visa	09/09	Trent Culpito	75 Wall St.	Ny, Ny 12212	212-552-1867	$19.99

2. Position your mouse pointer in the dataset editor, right-click, and then click **Add Record**. Click **OK** to add a row after the first row.
3. To add a second empty row, right-click **Add Record**.

   To save time, copy the data from row 0 in the dataset into the two empty rows that you created in steps 2 and 3.

4. Position the mouse pointer in the row 0 cell, right-click, and then click **Copy**.
5. Position the mouse pointer in row 1 cell, right-click, and then click **Paste**.
6. Click **Yes** to paste the data into the empty row.
7. Position the mouse pointer in row 2 cell, right-click, and then click **Paste**.
8. Click **Yes** to paste the data into the empty row.
9. Change the value in the **Quantity** and **Total** columns to test that ClassicsCD sample application correctly totals each order:
   a. In row 1, under the Quantity column, select the cell and type `2`.
   b. In row 1, under the Total column, select the cell and type `$38.98`.
   c. In row 2, under the Quantity column, select the cell and type `3`.
   d. In row 2, under the Total column, select the cell and type `$57.97`.

   The data in the dataset should look like the following table:

	Com-poser	Item	Quan-tity	Card#	Card-Type	Exp-Date	Name	Street	CityStZip	Phone	Total
0	Schu-bert	String Quar-tets Nos. 4 & 14	1	1234567890	Visa	09/09	Trent Culpi-to	75 Wall St.	Ny, Ny 12212	212-552-1867	$19.99
1	Schu-bert	String Quar-tets Nos.	2	1234567890	Visa	09/09	Trent Culpi-to	75 Wall St.	Ny, Ny 12212	212-552-1867	$19.99

	Com-poser	Item	Quan-tity	Card#	Card-Type	Exp-Date	Name	Street	CityStZip	Phone	Total
0	Schu-bert	String Quar-tets Nos. 4 & 14	1	1234567890	Visa	09/09	Trent Culpi-to	75 Wall St.	Ny, Ny 12212	212-552-1867	$19.99
1	Schu-bert	String Quar-tets Nos.	1	1234567890	Visa	09/09	Trent Culpi-to	75 Wall St.	Ny, Ny 12212	212-552-1867	$38.98

	Com-poser	Item	Quan-tity	Card#	Card-Type	Exp-Date	Name	Street	City	St	Zip	Phone	Total
		4 & 14											
2	Schu-bert	String Quar-tets Nos. 4 & 14	*3*	1234567890		09/09	Trent Culpi-to	75 Wall St.	Ny, Ny		12212	212-555-1967	$19.97

	Com-poser	Item	Quan-tity	Card#	Card-Type	Exp-Date	Name	Street	City	St	Zip	Phone	Total
		4 & 14											
0	Schu-bert	String Quar-tets Nos. 4 & 14	1	1234567890		09/09	Trent Culpi-to	75 Wall St.	Ny, Ny		12212	212-555-1967	$7.97

10. On the **Test dataset** tab, click **X** to close the dataset editor, and then click **Yes** to save the changes you made to the dataset.

Advance to Lesson 6: Play back the test on page 174.

Lesson 6: Play back the test

In this lesson, you will play back the test you just recorded and use a variety of data from a dataset to test the application.

Each time you play back a script with an associated dataset, the script accesses one record in the dataset. When you create a dataset reference for a verification point, the verification point uses the dataset reference to access a variable in that record. During playback, Rational® Functional TesterHCL OneTest™ UI substitutes the variable in the dataset for the dataset reference and compares the variable in the dataset to the actual results.

During playback you can view the script name, the script line number that is executing, status icons, and a description of the action in progress in the Playback Monitor.

1. To play back the test script, click **Script > Run**.

   The Select Log dialog box opens.

2. Click **Next**.
3. Click the **dataset iterator Count** arrow and scroll up to select **Iterate Until Done** to access all three records in the dataset.
4. Click **Finish** to use the default log name.

   The Rational® Functional TesterHCL OneTest™ UI window minimizes, and the Playback Monitor appears in the top right area of your screen. Messages appear in the Playback Monitor as Rational® Functional TesterHCL OneTest™ UI plays back all of the recorded actions in the test script and enters data from the dataset.

When the test script finishes playing back, Rational® Functional TesterHCL OneTest™ UI displays a log with the test results. A log is a file that contains the record of events that occur while playing back a script. A log includes the results of all verification points executed that can be used to test the application.

5. Click **X** to close the log.

This completes the tutorial. You have data-driven a test script, created descriptive headings for the data collected, added data to the dataset, created a data verification point with a dataset reference, played back a test script, and viewed the log.

To learn more about data-driving test scripts and the dataset editor, see the Data-Driving Tests section of the Rational® Functional TesterHCL OneTest™ UI Help.

Test Adobe Flex application

This tutorial walks you through the steps to enable your Adobe Flex application, and test the enabled Flex application from a local test computer using Rational® Functional TesterHCL OneTest™ UI. The steps are based on the roles that Flex developers and testers perform.

Rational® Functional TesterHCL OneTest™ UI supports testing the functional aspects of Flex applications. You can record and playback scripts against Flex based user interfaces inside a web browser.

Learning objectives

After completing this tutorial, you will be able to use Rational® Functional TesterHCL OneTest™ UI to test your Flex applications.

Exemple

Time required

30 minutes

Introduction: Test Adobe Flex application

In this tutorial, you will learn how to enable a Flex application for functional testing and test the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI.

The tutorial is divided into two modules that must be completed in sequence to work properly.

Learning objectives

After completing this tutorial, you will be able to:

- Set up the development and testing environment
- Enable the Flex application for functional testing
- Test the enabled Flex application using Rational® Functional TesterHCL OneTest™ UI

Note: Consider printing the tutorial before you begin and using the printed copy as you work through the lessons. You can print each lesson by right-clicking inside each topic and clicking **Print**.

Exemple

Time required

This tutorial takes approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Module 1: Enable the Flex application for testing

In this module, you learn how to enable Flex applications for functional testing by compiling the Flex application with functional testing agent (rft.swc) and Flex automation framework libraries.

This module is intended for Flex application developers.

Learning objectives

After you complete the lessons in this module you will understand how to do the following:

- Set up the development environment
- Configure the Flex application

Exemple

Time required

This module takes approximately 15 minutes to complete.

Exemple

Prerequisites

Before you begin, verify that the following software is installed in your computer:

- Adobe Flex SDK 4.5
- Adobe Flex automation framework

Lesson 1: Set up the development environment

In this lesson, you set up the development environment for enabling Flex application for functional testing.

About this task

To set up the Flex development environment:

1. Copy the automation_agent.swc file from `C:/Program Files/Adobe/frameworks/libs` directory to `C:/Program Files/Adobe/Flex SDK 4.5/frameworks/libs` directory.
2. Copy the automation_agent_rb.swc file from `C:/Program Files/Adobe/frameworks/locale/en_US` directory to `C:/Program Files/Adobe/Flex SDK 4.5/frameworks/locale/en_US` directory.



Important: This path is for en_US locale. If you are using a different locale, replace en_US with that locale.

Results

Now you are ready to configure the Flex application.

Lesson 2: Configure the Flex application

In this lesson, you use the Flex user interface to setup the configuration parameters for the Flex application.

About this task

To configure the Flex application:

1. Click **Configure > Configure Applications for Testing**.
2. Click **Add** in the Application Configuration Tool window.
3. Select **Flex Application**, and click **Next**.
4. Select **Configure Flex application setup**, and click **Next**.
5. Click **Local Application**.
6. Specify the Flex application parameters:
    a. Select **Flex 4.5** from Flex SDKs list.
    b. Select **Compile-time** from the Enablement type list.
    c. Click **Browse** to select the Flex application in .as or .mxml format.
    d. Select **Dependency Files** check box, if required.
    e. Click **Add** and browse to the dependency files.
    f. Select **Additional Libraries** check box.
    g. Click **Add** and browse to the libraries.
    h. Click **Browse** and select the SWF target location.
    i. Select the **Generate HTML Page** checkbox, and click **Finish**.

Results

The selected Flex application and its detailed information is displayed in the Application Configuration Tool window. Pass the generated HTML page to the tester to test the HTML page using Rational® Functional TesterHCL OneTest™ UI.

Module 2: Test the Flex application

In this module, you learn how to test the functionality of the enabled Flex application that is provided by the developer.

This module is intended for testers and describes the steps for testing the enabled Flex applications using Rational® Functional TesterHCL OneTest™ UI.

Learning objectives

After you complete the lessons in this module you will understand how to do the following tasks:

- Set up the testing environment
- Test the enabled Flex application from a local test computer

Exemple

Time required

This module takes approximately 15 minutes to complete.

Exemple

Prerequisites

Before you begin, verify that the following software is installed:

- Rational® Functional TesterHCL OneTest™ UI
- Supported browser
- Adobe Flash Player ActiveX control

Lesson 1: Assign trust designations

In this lesson, you assign trust designations to enable the file to be trusted.

About this task

To test the application from a local test computer, the application-under-test has to be a trusted application. The paths to individual files or directories can be assigned trust designations. This renders that all the files in each selected directory and any of its sub-directories are trusted.

To assign trust designations:

1. Create a folder named FlashPlayerTrust in `C:\WINDOWS\system32\Macromed\Flash`.
2. Create a file named Flex without any file extension.
3. Enter the path of the Flex application in the Flex file.
4. Save the file.

Results

Now you are ready to test the Flex application using the Rational® Functional TesterHCL OneTest™ UI.

Lesson 2: Test enabled Flex application from a local test computer

In this lesson, you test the functionality of the enabled Flex application that is embedded in an HTML wrapper on a local computer.

About this task

To test the enabled Flex application on a local computer:

1. Get the HTML wrapper from the developer.
2. Open the HTML page in a web browser.
3. Start Rational® Functional TesterHCL OneTest™ UI for testing the HTML application that contains the embedded Flex application.



Note: You can also test enabled Flex applications that are deployed on a web server. For more information, see the online help.

Chapter 5. Samples

You can find the sample project that you can use with Rational® Functional TesterHCL OneTest™ UI to test the functionality of an application.

Sample project to test a Java application

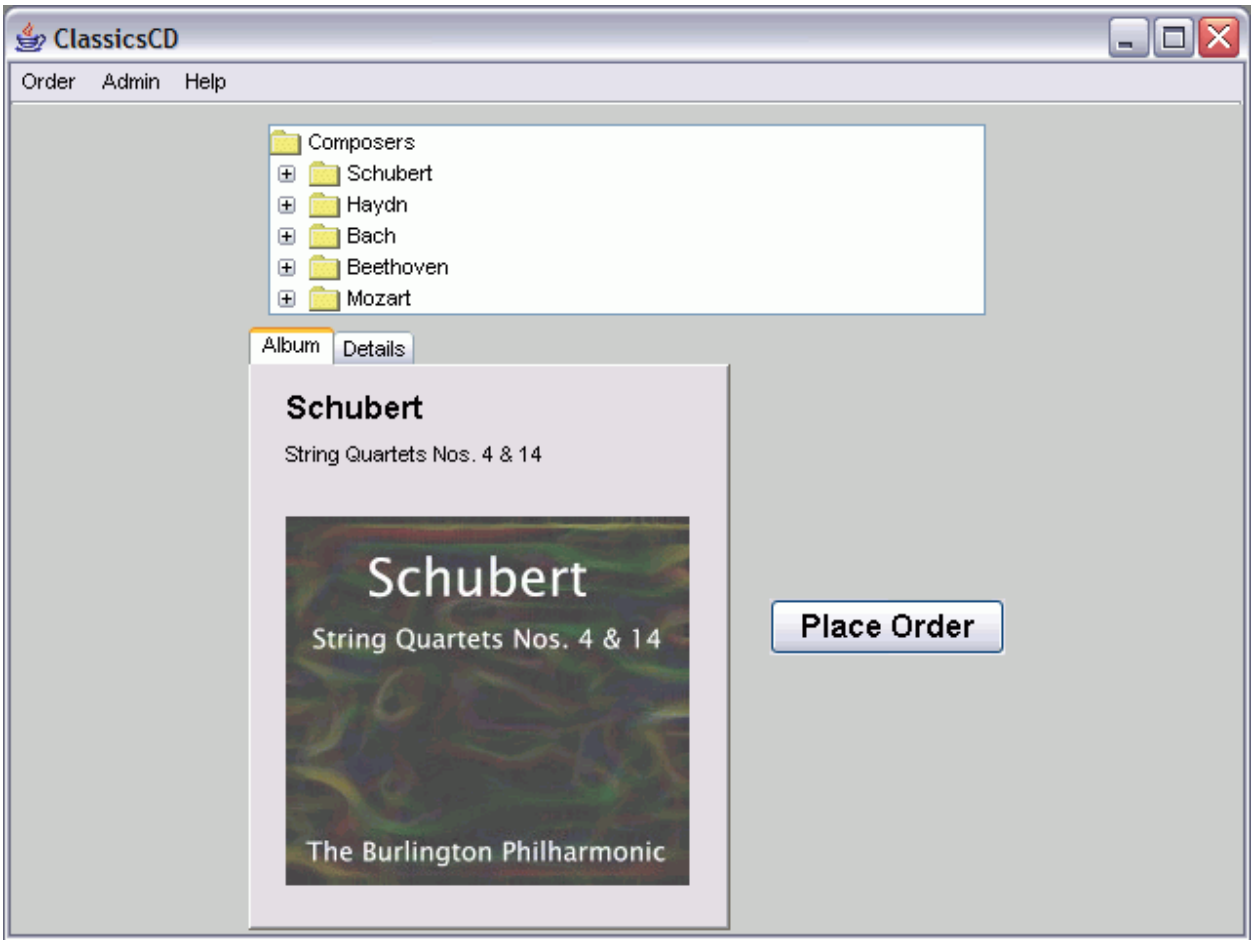
The sample functional test project is created using the first few lessons of the "Create a functional test" tutorial . You can look at the test assets while you do the tutorial or after you complete it.

Time required: 15 minutes

The sample project contains the script, verification points, object map, and other files that are created when you complete lesson 3 of the tutorial. This sample script does not contain the image verification point. For reference, you can look at the script or other assets to compare them with your own, or actually play back the script. The tutorial script is called ClassicsSample.

Click to enlarge image

The ClassicsCD sample application



Related reference

Task flow for testing Java applications on page 49

Related information

Get the sample

Functional testing of a Java application sample project on page 181

Create a functional test using Java scripts on page 104

Functional testing of a Java application sample project

To use these examples, copy the testobject and superscript directories and their contents into a functional test project.

To use one of the superscripts, set the helper superclass property for a script to the full class name of the superscript. For example, to use the `ExtensionScript` superclass, for a script called "X":

1. Right-click "X" in the Functional Test Projects View and then select **Properties** from the popup menu.
2. In the **Properties for X.java** window, select **Functional Test Script** in the list on the left.
3. Finally, set the text in the edit box labeled **Helper superclass** to `superscript.ExtensionScript`.

You can also modify your project preferences so that all newly created scripts in the project will extend this superscript. To define a default helper superscript for a project, perform these steps:

1. Right-click the project and then select **Properties** from the menu.
2. Then, set the text field in the Functional Test Project labeled **New Script Helper superclass**.

After completing these steps, your `X.java` script can make use of the additional methods of `ExtensionScript` such as `getClipboardText()`, `setClipboardText()`, `clipboardVP()`, etc.

Class	Package	Description
`ExtensionScript`	superscript	Provides some general utility methods.
`HtmlScript`	superscript	Provides a handler to automatically dismiss unexpected active HTML dialogs.
`WindowScript`	superscript	Provides some useful methods for getting around problems with native Microsoft® Windows® Applications.
`SwtScript`	superscript	Provides some useful methods for testing SWT-based applications. Note that this implementation makes use of WindowScript, which is Microsoft-Windows specific. This class will not work on Linux®.
`EclipseScript`	superscript	Provides some methods that may be useful when testing plugins running inside the Eclipse platform (see http://www.eclipse.org/). Note that this code makes use of internal Eclipse classes, and consequently might break with future versions of Eclipse. This class illustrates invoking static methods in the SUT and using custom test objects.
`WorkbenchTestObject`	testobject.eclipse	A test object for the Eclipse (see http://www.eclipse.org/) shell Workbench.
`WorkbenchWindowTestObject`	testobject.eclipse	A test object for the Eclipse (see http://www.eclipse.org/) shell WorkbenchWindow.

Class	Package	Description
`WorkbenchPageTestObject`	testobject.eclipse	A test object for the Eclipse (see http://www.eclipse.org/) shell WorkbenchPage.

A Rational® Functional TesterHCL OneTest™ UI project to test an HTML application

This is a sample functional test project that contains various functional test scripts for testing an HTML application. In this sample project, the IBM® Rational® Team Concert™ web client is used as an application under test.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Rational® Functional TesterHCL OneTest™ UI supports testing HTML applications that contain various types of controls. It also supports testing Dojo controls in an HTML application. In this sample project, the test scripts are for testing the function of an HTML application that contains various types of HTML controls such as text, check box, list, links and Dojo controls.

The sample functional test project contains simplified test scripts and the corresponding Java test scripts. After you import the project into your workspace, you can view the test scripts and the corresponding application visuals, the dataset, and the object maps.

Importing the sample

1. To import the sample into Rational® Functional TesterHCL OneTest™ UI workspace, click **Get the sample**.
2. Open the Functional Test perspective to view the imported functional test project.
3. To view both the simplified test script and the corresponding Java script, click **Windows > Preferences**.
4. Select the **Enable the simplified script** option.

Related reference

Task flow for testing HTML applications on page 51

Related information

Testing the sample on page 184

Get the sample

Download Rational Team Concert from jazz.net

Testing the sample

The functional test project contains simplified test scripts. You can view the simplified test scripts and the Java scripts.

Application under test: The IBM® Rational® Team Concert™ web client is used as an application under test.

To run the functional test script against the IBM® Rational® Team Concert™ web client, you must set up Rational® Team Concert™ for project-management activities.

If you do not have a set up for Rational® Team Concert™, you can obtain a trial version of Rational® Team Concert™ from Jazz.net. Register your details and download Rational® Team Concert™ from the Jazz.net web site, and set it up for project-management activities.

You can then, modify certain test lines in the functional test script according to your setup and run the sample test script against Rational® Team Concert™ to observe HTML testing.

This project contains the following two sample functional test scripts:

- **CreateDefect**: This functional test script creates a defect in Rational® Team Concert™. Some of the values that must be specified in the fields for creating a defect are data-driven. You can add values to the dataset, modify the script according to your setup, and run the script to test the application.
- **CreateQuery**: This functional test script creates a query that lists work items that match certain conditions in Rational® Team Concert™. Some of the values that must be specified in the fields for creating the query are data-driven. You can add values to the dataset, modify the script according to your setup, and run the script to test the application.

Functional testing proxy SDK technology samples

Functional testing proxy SDK samples are simple examples of proxy source code and applications under test (AUT).

These samples explain different implementations of Rational® Functional TesterHCL OneTest™ UI proxy APIs.

ButtonProxy

The `ButttonProxy` sample has two Rational® Functional TesterHCL OneTest™ UI proxies. There is one button control each for the Abstract Window Toolkit (AWT) and the Java foundation class (JFC). The sample works with the application-under-test (AUT) sample.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Importing samples

1. To import the sample into the Eclipse workspace, click Get the sample. You must import all the samples that are provided here into your workspace.
2. Open the Java perspective to view the imported samples.

   The proxy sample contains the following files:
   - Proxy source files
     - ButtonProxy\src\sdk\samples\awt\ExtendedButtonProxy.java
     - ButtonProxy\src\sdk\samples\jfc\ExtendedJButtonProxy.java
     - ButtonProxy\ButtonProxy.rftcust
   - Eclipse project files
     - ButtonProxy\.project
   - Proxy binary files
     - ButtonProxy\ButtonProxy.jar
     - ButtonProxy\ButtonProxy.rftcust

   The AUT sample contains the following files:
   - Eclipse project files
     - ButtonApp\.project
   - AWT button application-under-test files
     - ButtonApp\src\AWTButtonApp.java
     - ButtonApp\bin\AWTButtonApp.class
   - JFC or swing button application-under-test files
     - ButtonApp\src\JButtonApp.java
     - ButtonApp\bin\JButtonApp.class
3. Test the button proxy sample on page 185.

Related information

Get the proxy sample

Get the application sample

Testing the button proxy sample on page 185

Testing the button proxy sample

This proxy sample explains how to write a simple proxy, map proxies to controls, deploy proxies, and verify how proxies work.

Test the button application sample to view the default value of the button control

1. Open the `AWTButtonApp.java` and `JButtonApp.java` files that are available in the imported `ButtonApp` project folder.
2. Open the **Functional Test** perspective.
3. Run the `AWTButtonApp.java` script. The sample button application is displayed.

4. To test the button control, record a functional test script and click the button control of the sample application.
5. Open the test object map. Notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for java.awt.Button is `.java.awt.ButtonProxy`. This is the default value for this control.
6. Similarly run the `JButtonApp.java` script. The sample button application is displayed.
7. To test the button control, record a functional test script and click the button control of the sample application.
8. Open the test object map. Notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for javax.swing.JButton is `.java.jfc.AbstractButtonProxy`. This is the default value for this control.
9. Notice that the button click action is recorded as `button.Click()`

Deploy the binary files

1. Open the Java perspective.
2. From the ButtonProxy project, copy the `ButtonProxy.jar` and the `ButtonProxy.rftcust` to the customization directory. The default location for the customization directory is `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HCLOneTest\customization`.

Verify the proxy deployment

After deploying the proxy, you can now verify the value of the control.

1. Restart Rational® Functional TesterHCL OneTest™ UI
2. Open the `ButtonApp` application as mentioned in the earlier section.
3. Record a functional test script to test the buttons of the sample application.
4. Open the test object map. Notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for java.awt.Button and javax.swing.JButton are `sdk.sample.awt.ExtendedButtonProxy` and `sdk.sample.swt.ExtendedJButtonProxy` respectively. This proxy sample extends the proxy method `public String getDescriptiveName()` to change the TestObject descriptive names for the java.awt.Button and javx.swt.JButton controls.
5. Notice that after you deploy the proxies, the click actions on java.awt.Button and javax.swing.JButton controls are recorded as `button_button.click()` and `jbutton_button().click()` respectively as the proxy changes the descriptive name given to the TestObject for these two controls

JFormattedTextFieldProxy

This JFormattedTextFieldProxy sample explains how create a new proxy class add new control properties and control data. The sample works with the application-under-test (AUT) sample.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Importing samples

1. To import the sample into the Eclipse workspace, click Get the sample. You must import all the samples that are provided here into your workspace.
2. Open the Java perspective to view the imported samples.

   The proxy sample contains the following files:
   - Proxy source files
     - JFormattedTextFieldProxy\src\sdk\samples\jfc\JFormattedTextFieldProxy.java
     - JFormattedTextFieldProxy\JFormattedTextFieldProxy.rftcust
   - Eclipse project files
     - JFormattedTextFieldProxy\.project
   - Proxy binaries
     - JFormattedTextFieldProxy\JFormattedTextFieldProxy.jar
     - JFormattedTextFieldProxy\JFormattedTextFieldProxy.rftcust

   The AUT sample contains the following files:
   - Eclipse project files
     - JFormattedTextFieldApp\.project
   - Application-under-test files
     - JFormattedTextFieldApp\JFormattedTextFieldApp.java
     - JFormattedTextFieldApp\JFormattedTextFieldApp.class
3. Test the JFormattedTextField proxy sample on page 187.

Related information

Get the proxy sample

Get the application sample

Testing the sample on page 187

Testing the sample

This proxy sample explains how to extend a proxy to add more control properties and control data.

Test the application sample to view the default value of the text control

1. Open the `JFormattedTextFieldApp.java` file that is available in the imported `JFormattedTextFieldApp` project folder.
2. Open the **Functional Test** perspective.
3. Run the `JFormattedTextFieldApp.java` script. The sample application is displayed.
4. To test the text control, record a functional test script and record a data verification point and properties verification point on any of the control in the sample application.

5. Notice that there is no separate proxy for `javax.swing.JFormattedTextFieldProxy`. Properties that are specific to the `JFormattedTextFieldProxy` control, for example, format string and unformatted value are not available for the `getProperties()` method. These values are also unavailable for data verification points.
6. Run `testObject.getProperty("unformattedValue")`. This throws the error message, `Properties not found.`

Extended capabilities in the proxy code

Added more control properties

Along with the default control properties that are provided, more control properties are added by extending the `java.util.Hashtable getProperties()` and `Object getProperty(String propertyName)` proxy methods.

Added more control data

Along with the default control data types that are provided, more control data are added by extending the `java.util.Hashtable getTestDataTypes()` and `ITestData getTestData(String testDataType)` proxy methods.

Deploy the binary files

1. Open the Java perspective.
2. From the JFormattedTextFieldProxy project, copy the `JFormattedTextFieldProxy.jar` and the `JFormattedTextFieldProxy.rftcust` to the customization directory. The default location for the customization directory is `C:\ProgramData\IBM\RFT\customizationC:\ProgramData\IBM\HFT\customization`.

Verify the proxy deployment

1. Restart Rational® Functional TesterHCL OneTest™ UI
2. After you deploy the proxies, running `testObject.getProperty("unformattedValue")` returns a valid property.
3. Before you deploy the proxies, data verification on the javx.swt.JFormattedTextField control returns only two data types. After you deploy the proxies, an additional data type Unformatted Value is included. You can also verify that the additional data type is present by using the `getTestDataTypes()` and `getTestData("value")` APIs.

CheckBoxProxy

This CheckBoxProxy sample explains how to create a new proxy class and new TestObjects and then map them to the `javax.swing.JCheckBox` control. The sample works with the application-under-test (AUT) sample.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Importing samples

1. To import the sample into the Eclipse workspace, click Get the sample. You must import all the samples that are provided here into your workspace.
2. Open the Java perspective to view the imported samples.

The proxy sample contains these files:

- Proxy source files
  - CheckBoxProxy\src\sdk\sample\jfc\ExtendedCheckBoxProxy.java
  - CheckBoxProxy\CheckBoxProxy.rftcust
- TestObjects source files
  - ExtendedToggleGUITestObject\src\sdk\sample\ExtendedToggleGUITestObject.java (Java TestObject)
  - ExtendedToggleGUITestObject\SDK\Sample\ExtendedToggleGUITestObje ct.cs (.NET TestObject. See .NET CheckBox ProxySDK sample)
  - ExtendedToggleGUITestObject\ExtendedToggleGUITestObject.rftcust (Customization file for both Java and .NET TestObjects)
- Eclipse project files
  - CheckBoxProxy\.project
  - ExtendedToggleGUITestObject\.project
- Binary files
  - CheckBoxProxy\CheckBoxProxy.jar (Proxy Jar)
  - CheckBoxProxy\CheckBoxProxy.rftcust (Proxy Customization file)
  - ExtendedToggleGUITestObject\ExtendedToggleGUITestObject.jar (Java TestObject Jar)
  - ExtendedToggleGUITestObject\ExtendedToggleGUITestObject.dll (.NET TestObject assembly)
  - ExtendedToggleGUITestObject\ExtendedToggleGUITestObject.rftcust (Customization file for both Java and .NET TestObject)
- The AUT sample contains the following files:
  - Eclipse project files
    - CheckBoxApp\.project
  - Application-under-test files
    - CheckBoxApp\src\JCheckBoxApp.java
    - CheckBoxApp\bin\JCheckBoxApp.class

3. Test the check box proxy sample on page 189.

Related information

Get the proxy sample

Get the application sample

Get the ExtendedToggleGUITestObject proxy sample

Testing the CheckBoxProxy sample on page 189

Testing the CheckBoxProxy sample

This proxy sample explains how to create a simple proxy class and a new TestObject for a CheckBox control

Test the check box application sample to view the default value of the control

1. Open the `JCheckBoxApp.java` file that is available in the imported `CheckBoxApp` project folder.
2. Open the **Functional Test** perspective.
3. Run the `JCheckBoxApp.java` script. The sample application is displayed.
4. To test the button control, record a functional test script and click the button control of the sample application.
5. Open the test object map. Notice that the **Proxy Class Name (#proxy)** and **Test Object Class Name (#testobject)** property under **Administrative properties** for the check box is `.java.jfc.JCheckBoxProxy` and `ToggleGUITestObject` respectively. This is the default value for this control.
6. Also, notice that the `check()` and `uncheck()` methods are unavailable for the checkbox TestObject.

Extended Capabilities: Creating a new TestObject

In this sample proxy, a new TestObject is created and mapped to CheckBoxProxy proxy to add the `check()` and `uncheck()` methods. This proxy also extends the `public String getTestObjectClassName()` proxy method to return the canonical name of the newly created TestObject so that all javax.swing.JCheckBox controls have new TestObjects.

Note: You create a new TestObject only when you want the control to expose new methods that are not available in the existing Rational® Functional TesterHCL OneTest™ UI TestObject.

Deploy the binary files

1. Open the Java perspective.
2. From the CheckBoxProxy and ExtendedToggleGUITestObject projects, copy the `CheckBoxProxy.jar`, `CheckBoxProxy.rftcust`, `ExtendedToggleGUITestObject.jar`, and `ExtendedToggleGUITestObject.rftcust` to the customization directory. The default location for the customization directory is `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization`.

   Note: You must manually add the `ExtendedToggleGUITestObject.jar` file to the Rational® Functional TesterHCL OneTest™ UI project, if a compilation error is displayed in the Rational® Functional TesterHCL OneTest™ UI script for the check box control.

Deploy the binary files

Copy the `CheckBoxProxy.jar`, `CheckBoxProxy.rftcust`, `ExtendedToggleGUITestObject.jar`, and `ExtendedToggleGUITestObject.rftcust` files to the customization directory and restart Rational® Functional TesterHCL OneTest™ UI to test the sample application-under-test (AUT).

Verify the proxy and TestObject deployment

You can test the javax.swing.JCheckBox control, which the CheckBoxApp AUT provides.

1. Restart Rational® Functional TesterHCL OneTest™ UI
2. After you deploy the proxies, the administrative property values change to the following for the checkbox TestObject:

Administrative property	Value
Proxy Class Name (#proxy)	`SDK.Sample.ExtendedCheckBoxProxy`
Test Object Class Name (#testobject)	`ExtendedToggleGUITestObject`

3. After you deploy the proxies, the `check()` and `uncheck()` methods are available for the checkbox TestObject.

Button OverrideProxy

This ButtonOverrideProxy sample explains how you can extend the recording behavior of Rational® Functional TesterHCL OneTest™ UI, add more properties and set up a simple value manager and value class. This sample proxy is written for the `java.awt.Button` control. The sample works with the application-under-test sample (AUT).

Time required: 15 minutes

The proxy sample contains the following files:

- Proxy source files
  - ButtonOverrideProxy\src\sdk\sample\awt\ButtonOverrideProxy.java
  - ButtonOverrideProxy\src\sdk\sample\value\SimpleValue.java
  - ButtonOverrideProxy\src\sdk\sample\value\SimpleValueManager.java
  - ButtonOverrideProxy\ButtonOverrideProxy.rftcust
- Eclipse project files
  - ButtonOverrideProxy\.project
- Proxy binary files
  - ButtonOverrideProxy\ButtonOverrideProxy.jar
  - ButtonOverrideProxy\ButtonOverrideProxy.rftcust

The AUT sample contains the following files:

- Eclipse project files
  - ButtonApp\.project
- Application-under-test files
  - ButtonApp\src\AWTButtonApp.java
  - ButtonApp\bin\AWTButtonApp.class

Related information

Get the proxy sample

Get the application sample

Examples you can use on page 192

Extend Rational Functional TesterHCL OneTest UI capabilities using Proxy SDK on page 142

Examples you can use

This proxy sample explains how to extend a proxy to add more properties, set up a simple value class and value manager, and extend the recording behavior of Rational® Functional TesterHCL OneTest™ UI.

Extended Capabilities : Adding more properties

This proxy sample extends the proxy methods `public java.util.Hashtable getProperties()` and `public Object getProperty(String propertyName)` to add a new property `simpleValue`.

Note: It is not mandatory to have value classes and value managers for all additional properties.

Set up a simple value class and value manager

This proxy sample returns a user-defined data type (value class and value manager) as a return value for the property `simpleValue`.

Extend the record capability

This proxy sample extends the `public void processSingleMouseEvent(IMouseActionInfo action)` method to extend the recording behavior so that single clicks are recorded as `doubleclick()` methods and double-clicks are recorded as `click()` methods.

Deploy the binary files

Copy the `ButtonOverrideProxy.jar` and `ButtonOverrideProxy.rftcust` files to the customization directory and then restart Rational® Functional TesterHCL OneTest™ UI to test the sample application-under-test (AUT).

Verify the added property and value class

You can test the `java.awt.Button` and `javx.swt.JButton` controls, provided as part of the `AWTButtonApp` and `JButtonApp` AUTs.

- Before you deploy the proxies, run `testObject().getProperty("simpleValue")`. This throws the error message, `simpleValue is not a valid property.`
- After you deploy the proxies, run `testObject().getProperty("simpleValue")`. This returns the value, `FuBar`.

Verify the recording behavior

- Before you deploy the proxies, the java.awt.Button records single-clicks as `button.click()` and double-clicks as `button.doubleClick()`.
- After you deploy the proxies, the java.awt.Button records single-clicks as `button.doubleClick()` and double-clicks as `button.Click()`. The `click()` and `doubleClick()` methods are swapped.

JSpinnerProxy

This JSpinnerProxy sample explains how to extend recording behavior of a control with SubItems, and support the playback for the extended recording behavior. The sample works with the application-under-test (AUT) sample.

Time required: 15 minutes

The proxy sample contains the following files:

- Proxy source files
  - JSpinnerProxy\src\sdk\sample\jfc\JSpinnerProxy.java
  - JSpinnerProxy\JSpinnerProxy.rftcust
- Eclipse project files
  - JSpinnerProxy\.project
- Proxy binary files
  - JSpinnerProxy\JSpinnerProxy.jar (Proxy Jar)
  - JSpinnerProxy\JSpinnerProxy.rftcust (Proxy Customization file)

The AUT sample contains the following files:

- Eclipse project files
  - JSpinnerApp\.project
- Application-under-test files
  - JSpinnerApp\src\JSpinnerApp.java
  - JSpinnerApp\bin\JSpinnerApp.class

Related information

Get the proxy sample

Get the application sample

Examples you can use on page 193

Examples you can use

This proxy sample explains how to extend the recording behavior of a control with SubItems and support for the corresponding playback.

Extended Capabilities: Recording controls with SubItems

This proxy sample extends the `processSingleMouseEvent()` proxy API to modify the recording behavior of a spin control. Although a spin control has two buttons and one text control as its children, from a testing perspective it must be treated as a single control with no children appearing in the TestObject Map. (In the TestObjectMap, the buttons and the text are treated as SubItems.) The `processSingleMouseEvent()` implementation sets methods with suitable SubItems, for example `atButton("UP")` or `atButton("DOWN")` as parameters for recording. Also note that although there are child objects for the spin control, you must make sure they are not listed as separate TestObjects. Therefore, the `getChildAtPoint()` and `getChildren()` APIs are extended to return null values. These child objects are treated as SubItems.

Playback Support

To support playback for each SubItem that is introduced during recording, Rational® Functional TesterHCL OneTest™ UI looks for the screen rectangle for each SubItem through the proxy. The SubItem rectangle can be provided by extending `java.awt.Rectangle getScreenRectangle(Subitem subitem)` proxy API.

Deploy the binaries

Copy the `JSpinnerProxy.jar` and `JSpinnerProxy.rftcust` files to the customization directory and restart Rational® Functional TesterHCL OneTest™ UI to test the sample application-under-test (AUT).

Verify the recording behavior

You can verify the recording behavior of the spin control:

- Before you deploy the proxies, when you record results of the up and down buttons of JSpinner, the clicks are recorded as `button.click()`, where each up and down control is treated as a separate control.
- After you deploy the proxies, clicking on the up button is recorded as `spinner().click(atButton("UP"))`. Notice that the buttons are treated and recorded as SubItems and not as separate TestObjects.

Verify the playback behavior

- Before you deploy the proxies, playing back user actions with SubItems, for example `spinner().click(atButton("UP"))` throws an exception, `Point not found`.
- After you deploy the proxies, playing back user actions with SubItems work fine.

TreeProxy

This TreeProxy sample explains how to extend recording behavior of a control with SubItems, and support the playback for the extended recording behavior. The sample works with the application-under-test (AUT) sample. To view the samples, you must import the tree proxy, tree application and the tree custom control for which the proxy is being written into your workspace.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Importing samples

1. To import the sample into the Eclipse workspace, click Get the sample. You must import all the samples that are provided here into your workspace.
2. Open the Java perspective to view the imported samples.

   The proxy sample contains these files:
   - Proxy source files
     - `TreeProxy\src\sdk\sample\jfc\ExtendedJTreeProxy.java`
     - `TreeProxy\src\TreeProxy.rftcust`
   - Eclipse project files
     - `TreeProxy\.project`
   - Proxy binary files
     - `TreeProxy\TreeProxy.jar`
     - `TreeProxy\TreeProxy.rftcust`

   The application-under-test sample contains these files:
   - Eclipse project files
     - `TreeApp\.project`
   - Application-under-test files
     - `TreeApp\src\sdk\sample\jfc\CustomTreeSample.java.`
     - `TreeApp\bin\sdk\sample\jfc\CustomTreeSample.class`

   The tree custom control contains these files:
   - ExtendedTreeControl\src\sdk\sample\jfc\ExtendedJTree.java
3. Test the tree proxy sample on page 195.

Related information

Get the proxy sample

Get the application sample

Get the tree custom control

Testing the sample on page 195

Testing tree proxy sample

With this proxy sample you learn how to write a simple proxy, map proxies to controls (handled internally), deploy proxies, and verify that the proxies work.

Test the tree application sample to view the default value of the tree control

1. Open the `CustomTreeSample.java` file that is available in the imported TreeApp project folder.
2. Open the **Functional Test** perspective.
3. Run the `CustomTreeSample.java` script. The sample tree application is displayed.
4. To test the tree controls, record a functional test script and click the tree controls of the sample application.
5. Open the test object map. Notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for CustomTree is `.java.jfc.JTreeProxy`. This is the default value for this control.

Deploy the binary files

1. Open the Java perspective.
2. From the TreeProxy project, copy the `TreeProxy.jar` and the `TreeProxy.rftcust` to the customization directory. The default location for the customization directory is `C:\ProgramData\IBM\RFT \customization. C:\ProgramData\HCL\HOTUI\customization.`

Verify the proxy deployment

After deploying the proxy, you can now verify the value of the control.

1. Restart Rational® Functional TesterHCL OneTest™ UI
2. Open the `CustomTreeSample` application as mentioned in the earlier section.
3. Record a functional test script to test the tree controls of the sample application.
4. Open the test object map. Notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for CustomTree is `sdk.sample.jfc.ExtendedJTreeProxy`. This is the newly developed proxy for the CustomTree control.

Flex control proxy

This FlexCustomControl sample explains how to extend the recording behavior of a control with SubItems and support the playback for the extended recording behavior. The sample works with the application under test sample. To view the samples, import the flex proxy for which the proxy is being written into your workspace.

Time required: 15 minutes

Prerequisite: You must open the help from Rational® Functional TesterHCL OneTest™ UI to import the samples into your workspace.

Importing samples

1. To import the sample into the Eclipse workspace, click Get the sample. You must import all the samples that are provided here into your workspace.
2. Open the Java perspective to view the imported samples.

   The proxy sample contains these files:

- Proxy source files
  - `FlexCustomControl\src\sdk\sample\flex\FlexCustomControlProxy.java`
  - `FlexCustomControl\src\sdk\sample\flextestobject\FlexCustomControlTestObject.java`
  - `FlexCustomControl\src\FlexCustomControl.rftcust`
- Eclipse project files
  - `FlexCustomControl\.project`
- Proxy binary files
  - `FlexCustomControl\FlexCustomControl.jar`
  - `FlexCustomControl\FlexCustomControl.rftcust`

The application under test sample contains these files:

- Eclipse project file: `FlexCustomControlApp\.project`
- Application under test files: `FlexCustomControlApp\mypage.htm`

3. Test the flex proxy sample on page 197

Related information

Get the the flex custom control proxy sample

Get the flex custom control application sample

Testing the flex proxy sample on page 197

Testing the flex proxy sample

With this proxy sample you learn how to write a simple proxy, map proxies to controls (handled internally), deploy proxies, and verify that the proxies work.

To test the sample, complete these procedures:

1. Open the application file `mypage.htm` located at `<installation_folder>\SDP\FunctionalTester\Flex\flexcustomcontrolapp.zip` `<installation_folder>\HCLOneTest\FunctionalTester\Flex\flexcustomcontrolapp.zip` after unzipping it.
2. Add the sample application path to the trusted directory `C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust\` by creating a file without an extension. For example, create a file named `Flex` in the trusted directory and add the file path of the sample application `C:\Program Files\IBM\SDP\FunctionalTester\Flex\flexcustomcontrolapp\mypage.htm` `C:\Program Files\HCL\HCLOneTest\FunctionalTester\Flex\flexcustomcontrolapp\mypage.htm` to it.
3. Copy the contents of the `ClassInfo` tag in `C:\Program Files\IBM\SDP\FunctionalTester\Flex\flexcustomcontrolapp\FlexCustom.xml` `C:\Program Files\HCL\HCLOneTest\FunctionalTester\Flex\flexcustomcontrolapp\FlexCustom.xml` file into `<installation_folder>\bin\FlexEnv.xml` `<installation_folder>\bin\FlexEnv.xml` file.
4. Invoke the sample Flex application and try recording on it.

Deploy the binary files

Deploy the `FlexCustomControl.jar` and `FlexCustomControl.rftcust` files to the customization directory. For example, `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HCLOneTest\customization`. Close Rational® Functional TesterHCL OneTest™ UI, all Java enabled applications and browsers so that the new customization class is loaded. Restart Rational® Functional TesterHCL OneTest™ UI to test the sample application under test.

Verify the proxy deployment

You can test the controls that are provided as part of the FlexCustomControlApp application under test by verifying the proxy class name before and after deployment.

- Before you deploy the proxies, notice that the **Proxy Class Name (#proxy)** property under **Administrative properties** for FlexCustomControl is `.flex.FlexObjectProxy` . This is the default value for this control. Before you deploy the proxies, notice that the event method is captured as follows:

  ```
  flex_FlexCustomControl().performAction("Select","Food");
  ```

- After you deploy the proxies, `sdk.sample.flex.FlexCustomControlProxy` is the newly developed proxy for the FlexCustomControl control. After you deploy the proxies notice also that the event method is captured as follows:

  ```
  Select("Food");
  ```

Functional testing samples for the Visual Studio IDE

Rational® Functional TesterHCL OneTest™ UI ships with some example code you can use in your own scripts. You can open these examples from within the sample project shipped with Rational® Functional TesterHCL OneTest™ UI. The sample project was created using the Rational® Functional TesterHCL OneTest™ UI tutorial. For reference, you can look at the script and other test assets while you do the tutorial, or after you complete the tutorial, if you want to compare the script and test assets to your files or actually play back the script.

The sample project contains the script, verification points, object map, and other files that are created when you complete the tutorial. The sample script is in the state it would be in at the end of the tutorial. The tutorial script is named "ClassicsSample."

Opening the sample project

To open the examples in the sample project, click **File > Open > Project**, browse to C:\Program Files\IBM\SDP\FunctionalTester\vsnet\FTSampleProject, and open FTSampleProject.rftvs. The project displays in the Solution Explorer. Double-click on the script name in the Solution Explorer to display the script in the Code Editor window. To play back the script, click the Run Functional Test Script button  on the product toolbar.

Examples

The project contains example code you can use in your own scripts.

Class	Package	Description
ExtensionScript	superscript	Provides several general utility methods.
HtmlScript	superscript	Provides a handler to automatically dismiss unexpected active HTML dialog boxes.

Using the examples

If you want to use the examples, copy the superscript directories and their contents into a Functional Test project.

To use one of the superscripts:

1. Set the helper base class property for a script to the full class name of the superscript. For example, to use the ExtensionScript base class for a script called X, right-click on X in the Solution Explorer, and click Properties.
2. In the X Property Pages dialog box, select Functional Test Script in the list on the left.
3. In the Helper base class field, type superscript.ExtensionScript.
4. Click OK.

You can modify your project preferences to extend the ExtensionScript superscript for all new scripts in the project. To do so, you must define a default helper base class for a project.

To modify project preferences:

1. Right-click on the project and click Properties.
2. In the Rational® Functional TesterHCL OneTest™ UI Project field, type New Script Helper base class.
3. Click OK.

Now your X script can make use of the additional methods of ExtensionScript such as GetClipboardText(), SetClipboardText(), and ClipboardVP().

Functional testing proxy SDK samples for the Visual Studio IDE

Rational® Functional TesterHCL OneTest™ UI ships with some example proxy code you can use to learn about proxy SDK. You can open these examples from within the sample projects shipped with Rational® Functional TesterHCL OneTest™ UI.

The sample projects contain sample proxy code, customization files, and the applications under test (AUT).

Opening the sample projects

To open the examples in the sample proxy projects, click **File > Open > Project**, browse to C:\Program Files\IBM\SDP \FunctionalTester\vsnet\FTProxySamples, and open the Visual Studio .Net solution files with the .sln extension under different sample projects. The sample project displays in the Solution Explorer.

There are five sample projects that are shipped with Rational® Functional TesterHCL OneTest™ UI:

- Button
- CheckBox
- FormattedText
- SpinControl
- Grid

Note: After you open the above samples, perform the following steps:

1. Rebuild the proxy samples. The .dll file is generated in the proxy sample location. For example; `<product install directory>\SDP\FunctionalTester\vsnet\FTProxySamples \Button\ButtonOverrideProxy\bin\Debug`
2. Close IBM® Rational® Functional TesterHCL OneTest™ UI.
3. Copy the .dll file, and the .rftcust file from the proxy sample location to the customization directory available at `C:\Documents and Settings\All Users\Application Data\IBM\RFT \customization` location.

   Note: The .rftcust file is located in `<product install directory>\SDP \FunctionalTester\vsnet\FTProxySamples\<Proxy samples>` location.

4. Restart IBM® Rational® Functional TesterHCL OneTest™ UI.

Chapter 6. Administrator Guide

This guide describes how to install IBM® Rational® Functional TesterHCL OneTest™ UI software. After you install the software, you can perform administration tasks such as license configuration and integration with other products. This guide is intended for administrators.

Installation

You can find information about the installation of IBM® Rational® Functional TesterHCL OneTest™ UI.

Installation requirements

This section details hardware, software, and user privilege requirements that must be met in order to successfully install and run Rational® Functional TesterHCL OneTest™ UI.

You can also view the installation requirements for Rational® Functional TesterHCL OneTest™ UI, at www.ibm.com/software/awdtools/tester/functional/sysreq/index.html.

Hardware and Software requirements

Before you install the product, verify that your system meets the hardware and software requirements.

For a complete list of system requirements, see Rational® Functional Tester system requirementsHCL OneTest UI 10.1.0 system requirements.

User privileges requirements

You must have a user ID that meets the following requirements before you can install Rational® Functional TesterHCL OneTest™ UI.

- Your user ID must not contain double-byte characters.
- For Windows®, the user privileges required for installing depend on the version of Windows® on your computer:
    - **For Microsoft® Windows®**, you must log in to the Administrator account (or run as Administrator; right-click the program file or shortcut and select **Run as Administrator**) to perform the following tasks:
        - Install or update Installation Manager
        - Install or update a product offering
        - Install a license key for your product by using Installation Manager
- For Linux®: You must be able to log in as root.

Note: On Ubuntu, you must ensure that the environment variables that are set while installing the products are retained when you open Rational® Functional TesterHCL OneTest™ UI and the application-under-test.

- If you log on as a non-administrator user, you can run Rational® Functional TesterHCL OneTest™ UI and also test applications. Administrator access is only required for installing Rational® Functional TesterHCL OneTest™ UI.

Installing the product using Installation Manager

In this section, you will learn the pre-installation, installation, and post-installation tasks to install the product by using Installation Manager.

Installation terminology

Understanding these terms can help you take full advantage of the installation information and your Rational® Functional TesterHCL OneTest™ UI.

These terms are used in the installation topics.

Installation directory

The location of product artifacts after the package is installed.

Package

An installable unit of a software product. Software product packages are separately installable units that can operate independently from other packages of that software product.

Package group

A package group is a directory in which different product packages share resources with other packages in the same group. When you install a package using Installation Manager, you can create a new package group or install the packages into an existing package group. (Some packages cannot share a package group, in which case the option to use an existing package group is unavailable.)

Repository

A storage area where packages are available for download. A repository can be disc media, a folder on a local hard disk, or a server or web location.

Shared directory

In some instances, product packages can share resources. These resources are located in a directory that the packages share.

Planning the installation

Read all the topics in this section before you begin to install or update any of the product features. Effective planning and an understanding of the key aspects of the installation process can help ensure a successful installation.

Installation roadmap

The installation roadmap lists the high-level steps for installing your product.

Roadmap for installing Rational® Functional TesterHCL OneTest™ UI

Perform these tasks to install Rational® Functional TesterHCL OneTest™ UI:

1. Verify that your computer meets the minimum hardware and software requirements for installing the product. on page 201
2. Verify that your user ID meets the requirements for installing the product. on page 201
3. Review the rest of the planning information on page 204.
4. Complete any necessary pre-installation tasks. on page 210
5. Install the product. on page 212.

   Note:
   - Rational® Functional TesterHCL OneTest™ UI is also included in the Rational® Test WorkbenchHCL OneTest™ Studio package. You can use the Rational® Functional TesterHCL OneTest™ UI license when you install Rational® Functional TesterHCL OneTest™ UI through the Rational® Test WorkbenchHCL OneTest™ Studio package.
   - To install Rational® Functional TesterHCL OneTest™ UI on multiple computers, you can use the silent installation mechanism to install the package quickly. For information, see Installing silently topics in the Installation Manager information center

6. Perform any necessary post-installation tasks on page 225, such as configuring the appropriate product license and the help content.

Installation Manager overview

Installation Manager is a program for installing, updating, and modifying packages. It helps you manage the applications, or packages, that it installs on your computer. Installation Manager does more than install packages: It helps you keep track of what you have installed, determine what is available for you to install, and organize installation directories.

Installation Manager provides tools that help you keep packages up to date, modify packages, manage the licenses for your packages, and uninstallation packages.

You can download the most recent version of Installation Manager from Installation Manager downloads. Installation Manager is required to install each HCL product. You will need to create an account on jazz.net before you can download the software.

Installation Manager includes six wizards that make it easy to maintain packages:

- The **Install** wizard walks you through the installation process. You can install a package by simply accepting the defaults or you can modify the default settings to create a custom installation. Before you install, you get a complete summary of your selections throughout the wizard. Using the wizard you can install one or more packages at one time.
- The **Update** wizard searches for available updates to packages that you have installed. An update might be a released fix, a new feature, or a new version of the product. Details of the contents of the update are provided in the wizard. You can choose whether to apply an update.
- The **Modify** wizard helps you modify certain elements of a package that you have already installed. During the first installation of the package, you select the features that you want to install. Later, if you require other features, you can use the modify packages wizard to add them to your package. You can also remove features and add or remove languages.
- The **Manage Licenses** wizard helps you set up the licenses for your packages. Use this wizard to change your trial license to a full license, to set up your servers for floating licenses, and to select which type of license to use for each package.
- The **Roll Back** wizard helps you to revert to a previous version of a package.
- The **Uninstall** wizard removes a package from your computer. You can uninstall more than one package at a time.

For more information about Installation Manager, visit the Installation Manager help.

Installation considerations

Part of planning entails making decisions about installation locations, working with other applications, extending Eclipse, upgrading, migrating, and configuring help content.

Planning what features to install

You can customize your software product by selecting which features of Rational® Functional TesterHCL OneTest™ UI to install.

When you install the Rational® Functional TesterHCL OneTest™ UI product package by using Installation Manager, the installation wizard displays the features in the available product package. From the features list, you can select which to install. A default set of features is selected for you (including any required features). Installation Manager automatically enforces any dependencies between features and prevents you from clearing any required features.

Note: After you finish installing the package, you can still add or remove features from your software product by running the Modify Packages wizard in Installation Manager. See Modifying installations on page 228 for more information.

Rational® Functional TesterHCL OneTest™ UI installation features

You can choose to install some of the features of Rational® Functional TesterHCL OneTest™ UI. Some features are selected for installation by default. If a feature already exists in your shared resources directory, you cannot install that feature again.

Feature	Description	Selected for installation by default
Select Edition (Eclipse IDE)	Provides an automated functional and regression testing of Windows®, .NET, Java™, HTML 5, SAP, AJAX, PowerBuilder, Flex, Dojo, GEF, Visual Basic applications, Adobe® PDF documents, and zSeries®, iSeries®, and pSeries® applications through the Eclipse IDE. You can install a full licensed edition, which is available for Web UI testing. The .NET Framework is required for testing SAP applications.	Yes
IBM® Rational® Functional Tester-HCL OneTest™ UI Extension for Selenium/Appium	Provides the ability to run Selenium/Appium tests.	Yes
Microsoft® Visual Studio .NET Integration	Provides components to enable Visual Basic .NET scripting through the Microsoft Visual Studio 2015, 2017, or 2019 integrated development environment (IDE). **Note:** Microsoft Visual Studio integration for Rational® Functional TesterHCL OneTest™ UI might not get installed if the mandatory prerequisites are not installed, or if Microsoft Visual Studio is not installed properly. For more information, see Unable to install Rational® Functional TesterHCL OneTest™ UI Extension for Microsoft Visual Studio 2015, 2017 & 2019.	No
IBM® Rational® Test WorkbenchHCL OneTest™ Studio Extension for SAP Solutions	Supports extended functional testing of SAP GUI for Windows and SAP GUI for HTML.	Yes

Feature	Description	Selected for installation by default
	With the installation of this extension, you can record and play back scripts for the SAP UI with SAP controls and customized data verification of SAP controls.	
Java 8 OpenJDK with Eclipse OpenJ9	Provides the Java 8.0 binaries required for Rational® Functional TesterHCL OneTest™ UI. **Notes:** • If you want to shell share Rational® Functional TesterHCL OneTest™ UI along with other products that utilize OpenJDK, you must select **Java 8 OpenJDK with Eclipse OpenJ9** for only one of the shell shared products. You must not install multiple instances of OpenJDK in the same package group.	Yes
Jazz Eclipse Client for Rational Team Concert/Rational Quality Manager	Manages functional test assets using the Jazz source control management provided by the Jazz Eclipse Client. You must have a compatible version of Rational Team Concert or Rational Quality Manager server setup to use the Jazz source control management. You must also have a platform	No

Feature	Description	Selected for installation by default
	that is also supported by the Rational Team Concert client.	

Note: Rational® Functional TesterHCL OneTest™ UI supports integration with Rational® Quality Manager for remote execution of test scripts. You can enable Rational® Functional TesterHCL OneTest™ UI integration with Rational® Quality Manager by configuring the adapter that is installed by default when you install Rational® Functional TesterHCL OneTest™ UI. For more information, see Rational Quality Manager integration overview.

Extending an existing Eclipse IDE

When you install the Rational® Functional TesterHCL OneTest™ UI product package, you can choose to extend an Eclipse integrated development environment (IDE) already installed on your computer by adding the functions that the Rational® Functional TesterHCL OneTest™ UI package contains.

The Rational® Functional TesterHCL OneTest™ UI package that you install using IBM® Installation Manager is bundled with a version of the Eclipse IDE or workbench; this bundled workbench is the base platform for providing the functionality in the Installation Manager package. However, if you have an existing Eclipse IDE on your workstation, then you have the option to *extend* it, that is, add to the IDE the additional functionality provided in the Rational® Functional TesterHCL OneTest™ UI package.

To extend an existing Eclipse IDE, in the Location page of the Install Packages wizard, select the **Extend an existing Eclipse IDE** option.

Important: To enable users who do not have Administrator privileges to work with Rational® Functional TesterHCL OneTest™ UI in the Windows® operating system, do not install Eclipse inside the Program Files directory (C:\Program Files\).

You might extend your existing Eclipse IDE, for example, because you want to gain the functionality provided in the Rational® Functional TesterHCL OneTest™ UI package, but you also want to have the preferences and settings in your current IDE when you work with the functionality from the Rational® Functional TesterHCL OneTest™ UI package. You also might want to work with plug-ins that you have installed that already extend the Eclipse IDE.

Your existing Eclipse IDE must be version 4.6.3 for the latest updates from eclipse.org to be extended. Installation Manager checks that the Eclipse instance you specify meets the requirements for the installation package.

Note: You might need to update your Eclipse version in order to install updates to Rational® Functional TesterHCL OneTest™ UI. Refer to the update release documentation for information on changes to the prerequisite Eclipse version.

Extending an existing Eclipse IDE

When you install the Rational® Functional TesterHCL OneTest™ UI product package, you can choose to extend an Eclipse integrated development environment (IDE) already installed on your computer by adding the functions that the Rational® Functional TesterHCL OneTest™ UI package contains.

The Rational® Functional TesterHCL OneTest™ UI package that you install using IBM® Installation Manager is bundled with a version of the Eclipse IDE or workbench; this bundled workbench is the base platform for providing the functionality in the Installation Manager package. However, if you have an existing Eclipse IDE on your workstation, then you have the option to *extend* it, that is, add to the IDE the additional functionality provided in the Rational® Functional TesterHCL OneTest™ UI package.

To extend an existing Eclipse IDE, in the Location page of the Install Packages wizard, select the **Extend an existing Eclipse IDE** option.

Important: To enable users who do not have Administrator privileges to work with Rational® Functional TesterHCL OneTest™ UI in the Windows® operating system, do not install Eclipse inside the Program Files directory (C:\Program Files\).

You might extend your existing Eclipse IDE, for example, because you want to gain the functionality provided in the Rational® Functional TesterHCL OneTest™ UI package, but you also want to have the preferences and settings in your current IDE when you work with the functionality from the Rational® Functional TesterHCL OneTest™ UI package. You also might want to work with plug-ins that you have installed that already extend the Eclipse IDE.

Your existing Eclipse IDE must be version 4.6.3 for the latest updates from eclipse.org to be extended. Installation Manager checks that the Eclipse instance you specify meets the requirements for the installation package.

Note: You might need to update your Eclipse version in order to install updates to Rational® Functional TesterHCL OneTest™ UI. Refer to the update release documentation for information on changes to the prerequisite Eclipse version.

Installation repositories

Installation Manager retrieves product packages from specified repository locations.

If the launchpad is used to start Installation Manager, the repository information is passed to Installation Manager.

If the Installation Manager is started directly, you must specify an installation repository that contains the product packages that you want to install. See Setting repository preferences in Installation Manager on page 209.

Some organizations bundle and host their own product packages on their intranet. Your system administrators will need to provide you with the correct URL.

By default, Installation Manager uses an embedded URL in each product to connect to a repository server over the Internet. Installation Manager then searches for the product packages as well as new features.

Setting repository preferences in Installation Manager

When you start the installation of Rational® Functional TesterHCL OneTest™ UI from the launchpad program, the location of the repository that contains the product package you are installing is automatically defined in IBM® Installation Manager when it starts.When you download and install Installation Manager separately, you must specify the repository preference (the URL for the directory that contains the product package) in Installation Manager before you can install the product package.

Before you begin

Specify these repository locations on the Repositories page of the Preferences window. By default, Installation Manager uses an embedded URL in each Rational® software development product to connect to a repository server through the Internet and search for installable packages and new features. Your organization may require you to redirect the repository to use intranet sites.

Before starting the installation process, be sure to obtain the installation package repository URL from your administrator.

To add, edit, or remove a repository location in Installation Manager:

1. Start Installation Manager.
2. On the Start page of Installation Manager, click **File > Preferences**, and then click **Repositories**.
   Result
   The Repositories page opens, showing any available repositories, their locations, and whether they are accessible.
3. On the **Repositories** page, click **Add Repository**.
4. In the **Add repository** window, type the URL of the repository location or browse to it and set a file path.
5. Click **OK**. If you provided an HTTPS or restricted FTP repository location, then you will be prompted to enter a user ID and password.
   Result
   The new or changed repository location is listed. If the repository is not accessible, a red x is displayed in the **Accessible** column.
6. Click **OK** to exit.

What to do next

Note: For Installation Manager to search the default repository locations for the installed packages, ensure the preference **Search service repositories during installation and updates** on the Repositories preference page is selected. This preference is selected by default.

Preinstallation tasks

Before you install the product, you might need to prepare or configure your computer.

Increasing the number of file handles on Linux workstations

About this task

Important: For best results, increase the number of file handles available for Rational® Functional TesterHCL OneTest™ UI, because it uses more than the default limit of 1024 file handles per process. (A system administrator might need to make this change.)

Exercise caution when following these steps to increase your file descriptors on Linux®. Failure to follow the instructions correctly might result in a computer that will not start correctly. For best results, have your system administrator perform this procedure.

1. Log in as root. If you do not have root access you will need to obtain it before continuing.
2. Change to the etc directory
3. Use the vi editor to edit the initscript file in the etc directory. If this file does not exist, type `vi initscript` to create it.

   Important: If you decide to increase the number of file handles, **do not** leave an empty initscript file on your computer. If you do so, your machine will not start up the next time that you turn it on or restart.
4. On the first line, type `ulimit -n 4096` (the key here is that the number is significantly larger than 1024, the default on most Linux computers). **Caution:** do not set this too high, because it can seriously impact system-wide performance.
5. On the second line, type `eval exec "$4"`.
6. Save and close the file after making sure you have done steps 4 and 5.

   Note: Ensure you have followed the steps correctly, as not doing this correctly will result in a machine that does not boot.

7. **Optional:** Restrict your users or groups by modifying the limits.conf file in the etc/security directory. Both SUSE Linux Enterprise Server (SLES) Version 9 and Red Hat Enterprise Linux Version 4.0 have this file by default. If you do not have this file, you might consider a smaller number in step 4 above (for example, 2048). You need to do this so that most users have a reasonably low limit on the number of allowable open files per process. If you used a relatively low number in step 4, it is less important to do this. However, if you choose to set a high number in step 4, refraining from establishing limits in the limits.conf file can seriously impact computer performance.

The following is a sample limits.conf file that restricts all users and then sets different limits for others afterwards. This sample assumes you set descriptors to 8192 in step 4 earlier.

```
*      soft nofile 1024
*      hard nofile 2048
root    soft nofile 4096
root    hard nofile 8192
user1   soft nofile 2048
user1 hard nofile 2048
```

Note that the ***** in the example above sets the limits for all users first. These limits are lower than the limits that follow. The root user has a higher number of allowable descriptors open, while user1 is in between the two. Make sure you read and understand the documentation contained within the limits.conf file before making your modifications.

What to do next

For more information on the ulimit command, refer to the man page for ulimit.

Verifying and extracting electronic images

After you download the installation files, you must extract the electronic image from the compressed file before you can install Rational® Functional TesterHCL OneTest™ UI. You may want to verify the completeness of the downloaded files before extracting the image.

About this task

If you select the Download Director option for downloading the installation file, the Download Director applet automatically verifies the completeness of each file that it processes.

Extracting the downloaded files

About this task

Extract each compressed file to the same directory. For Linux®: Do not include spaces in the directory names, or you won't be able to run the launchpad.sh command to start the launchpad from a command line.

Installing software

This section provides the instructions for installing Installation Manager and the product as well as installation verification.

To install your product, follow the procedures and information in these topics.



Note: Rational® Functional TesterHCL OneTest™ UI is also included in the Rational® Test WorkbenchHCL OneTest™ Studio package. You can use the Rational® Functional TesterHCL OneTest™ UI license when you

install Rational® Functional TesterHCL OneTest™ UI through the Rational® Test WorkbenchHCL OneTest™ Studio package.

Starting the launchpad

You can use the launchpad program that provides you with a single location to start the installation process.

About this task

Use the launchpad program to start the installation of Rational® Functional TesterHCL OneTest™ UI in these cases:

- Installing from an electronic image on your local file system
- Installing from an electronic image on a shared drive

By starting the installation process from the launchpad program, IBM® Installation Manager is automatically installed if it is not already on your computer. Furthermore, the installation process is already configured with the location of the repository that contains the installation package. If you install and start Installation Manager directly, then you must set repository preferences manually.

To install the product from electronic disks downloaded from IBM Passport Advantage, open a command line and change to the directory where you extracted the disk images, and then do one of the following things:

- On Windows, run `RFT_SETUP\launchpad.exe`.
- On Linux, go to the `RFT_SETUP` directory and type `./launchpad.sh` to start the launchpad.

Installing Rational® Functional TesterHCL OneTest™ UI

To get started, you must install Rational® Functional TesterHCL OneTest™ UI.

Before you begin

- You must have obtained the installation media or downloaded the installation package that is available on the IBM Passport Advantage site.
- Your computer must have met the system requirements and completed the preinstallation tasks.
- To enable the Installation Manager to search for the latest available version of the packages in the predefined IBM update repository, your computer must be connected to the Internet.
- If you want to install the IBM® Rational® Functional TesterHCL OneTest™ UI on Linux machine, the following libraries must be available on your machine.
    - libXp.so.6
    - libXm.so.4
- To install Rational® Functional TesterHCL OneTest™ UI on Red Hat Linux Enterprise (RHEL) 8.0 and later, you must have completed one of the following tasks:
    - Installed libnsl.so.1 on your computer
    - Created a soft link to libnsl.so.2 by running the following command:sudo ln -s /usr/lib64/libnsl.so.2 / usr/lib64/libnsl.so

- To install Rational® Functional TesterHCL OneTest™ UI on a computer with Ubuntu operating system, you must have installed `libxm4` and `libxp6` packages.
    - Add deb http://security.ubuntu.com/ubuntu precise-security main to `/etc/apt/sources.list`
    - From the terminal, run these commands:

      $ sudo apt-get update and $ sudo apt-get install libxm4 libxp6

    Note: You must install Rational® Functional TesterHCL OneTest™ UI using either root user or Sudo access.

- To install Rational® Functional TesterHCL OneTest™ UI on the Windows® operating system and integrate it with Microsoft Visual Studio IDE, you must have installed the correct version of Microsoft Visual Studio.

    Note: Microsoft Visual Studio integration for Rational® Functional TesterHCL OneTest™ UI might not get installed if the mandatory prerequisites are not installed, or if Microsoft Visual Studio is not installed properly. For more information, see Unable to install Rational® Functional TesterHCL OneTest™ UI Extension for Microsoft Visual Studio 2015, 2017 & 2019.

- You cannot upgrade to the latest version of the product. You must uninstall the existing version of the product before installing the latest version of the product.

- HCL Software manages software licensing and downloads through the License and Delivery Portal. You must have ordered software and followed the instructions that you received to activate your entitlement and download the software from the Portal. These instructions help you complete the following steps:
    - Order the software. Gather the URL and login credentials for the HCL® License & Delivery portal.
    - Log in to the Portal.
    - Create a User associated with your account.
    - Create a Device to host the license server and record the server ID that is returned when the device is created.

      Important: The server ID uniquely identifies your company's entitlement and must be kept secure.

    - Map your entitlement to the license server Device.
    - Download the software.
- Download and install Installation Manager.
- To install the product on a Windows computer, install Oracle JDK 8.
- If you were using the IBM testing product and want to reuse the test assets in the HCL testing product, follow these instructions:
    - From the IBM testing product, export all the test assets with dependencies.
    - Uninstall the IBM testing product.

- Install the HCL testing product by following the instructions in this topic.
- Import the test assets with dependencies to the HCL workspace.

Perform the following steps to install Rational® Functional TesterHCL OneTest™ UI on Windows® or Linux® operating systems:

1. If you are using the installation media, insert the media into the disk drive. If you are using the download package, extract the files from the downloaded .zip package to a temporary directory.
2. Open the launchpad program to start the package installation:
   Choose from:
   - On Windows: Click **launchpad.exe**.
   - On Linux: Click **launchpad.sh**.
3. Click **Install Rational Functional Tester**.
4. Start Installation Manager and from **File > Preferences > Repositories**, add a new repository link by pointing to the setup disk of the product.
5. On the Installation Manager Start page, click **Install**.

   The Install Packages page lists all the packages found in the repositories that Installation Manager searched. If two versions of a package are discovered, only the most recent, or recommended version of the package is displayed. If Installation Manager is not installed on your computer, it is listed on the Install Packages page with Rational® Functional TesterHCL OneTest™ UI.

   a. To display all versions of the packages that Installation Manager finds, click **Show all versions**.

      Note: To search the predefined IBM update repository locations for the installed packages, ensure that **Search service repositories during installation and updates** check box is selected on the Repositories preference page. This preference is selected by default.

      If updates for the Rational® Functional TesterHCL OneTest™ UI package are found, then they are displayed on the **Installation Packages** list on the Install Packages page. The latest updates are displayed by default.

   b. To find other versions, fixes and extensions for the available packages in the update repository locations, click **Check for Other Versions, Fixes and Extensions**.

   c. To view all updates found for the available packages, click **Show all versions**.

   d. To display a package description under **Details**, click the package name. If additional information about the package is available, such as a readme file or release notes, a **More info** link is included at the end of the description text. To fully understand the package you are installing, review all information beforehand.

6. Select **Rational Functional TesterHCL OneTest UI**, the required version, and IBM® Installation Manager, if necessary and click **Next**.

Updates that have dependencies are automatically selected and cleared together.

Note: If you install multiple packages simultaneously, then all the packages will be installed into the same package group.

7. On the Licenses page, read the license agreement for the selected package.

   If you select multiple packages to install, each package might contain a license agreement. On the left side of the **License** page, click each package version to display its license agreement. The package versions that you selected to install (for example, the base package and update) are listed under the package name.

   a. If you agree to the terms of all of the license agreements, click **I accept the terms of the license agreements**.
   b. Click **Next** to continue.

Note: Rational® Functional TesterHCL OneTest™ UI is also included in the Rational® Test WorkbenchHCL OneTest™ Studio package. You can use the Rational® Functional TesterHCL OneTest™ UI license when you install Rational® Functional TesterHCL OneTest™ UI through the Rational® Test WorkbenchHCL OneTest™ Studio package. However, if Rational® Test WorkbenchHCL OneTest™ Studio is installed on Microsoft® Windows® 2012, you cannot install and use Rational® Functional TesterHCL OneTest™ UI on the same operating system.

8. On the Location page, type the path for the *shared resources directory* in the **Shared Resources Directory** field, or accept the default path. The shared resources directory contains resources that can be shared by one or more package groups. Click **Next** to continue.

Important: If you are installing Rational® Functional TesterHCL OneTest™ UI on Windows®, and do not have Windows® Administrator privileges to work with Rational® Functional TesterHCL OneTest™ UI, you must not choose a directory inside the Program Files directory (C:\Program Files\).

   The default path is:
   - For Windows®: `C:\Program Files\IBM\IBMIMShared C:\Program Files\HCL\HCLIMShared`
   - For Linux®:`/opt/IBM/IBMIMShared /opt/HCL/HCLIMShared`

Important: You can specify the shared resources directory only the first time when you install a package. You must use your largest disk partition to ensure adequate disk space is available for the shared resources of future packages. You cannot change the directory location unless you uninstall all packages.

9. Also, on the Location page, specify whether to create a *package group* and install the IBM® Rational® Functional TesterHCL OneTest™ UI package into a new package group or use an existing package group to

shell-share with another offering. A package group represents a directory in which packages share resources with other packages in the same group. To create a new package group:

a. Click **Create a new package group**.

b. Type the path for the installation directory for the package group.
The name for the package group is created automatically.

The default path is as follows:
- For Windows®: `C:\Program Files\IBM\SDP C:\Program Files\HCL\HCLOneTest`
- For Linux®: `/opt/IBM/SDP/opt/HCL/HCLOneTest`

Note: If you are installing Rational® Functional TesterHCL OneTest™ UI on a Linux® machine, ensure that you do not include any spaces in the directory path.

c. Click **Next** to continue.

10. On the next Location page, you can choose to extend an existing Eclipse IDE already installed on your system, adding the functionality in the packages that you are installing.

Note: You must have Eclipse 3.6.2 with the latest updates from eclipse.org to select this option.

Choose from:
- If you do not want to extend an existing Eclipse IDE, click **Next** to continue.
- To extend an existing Eclipse IDE:
  a. Select **Extend an existing Eclipse**.
  b. In the **Eclipse IDE** field, type or navigate to the location of the folder containing the eclipse executable file (eclipse.exe or eclipse.bin). Installation Manager checks whether the Eclipse IDE version is valid for the package that you are installing. The **Eclipse IDE JVM** field displays the Java™ Virtual Machine (JVM) for the IDE that you specified.
  c. Click **Next** to continue.

11. On the Features page under **Languages**, select the languages for the package group, and then click **Next**.

The corresponding national language translations for the user interface and documentation for the Rational® Functional TesterHCL OneTest™ UI package is installed.

12. Select the package features that you want to install.

a. Select or clear features in the packages. Installation Manager automatically enforces any dependencies with other features and displays the updated download size and disk space requirements for the installation.

Feature	Description
Select Edition (Eclipse IDE)	Provides an automated functional and regression testing of Windows®, .NET, Java™, HTML 5, SAP, AJAX, PowerBuilder, Flex, Dojo, GEF, Visual Basic applications, Adobe® PDF documents, and zSeries®, iSeries®, and pSeries® applications through the Eclipse IDE. You can install a full licensed edition, which is available for Web UI testing. The .NET Framework is required for testing SAP applications.
IBM® Rational® Functional TesterHCL OneTest™ UI Extension for Selenium/Appium	Provides the ability to run Selenium/Appium tests.
Microsoft® Visual Studio .NET Integration	Provides components to enable Visual Basic .NET scripting through the Microsoft Visual Studio 2015, 2017, or 2019 integrated development environment (IDE). **Note:** Microsoft Visual Studio integration for Rational® Functional TesterHCL OneTest™ UI might not get installed if the mandatory prerequisites are not installed, or if Microsoft Visual Studio is not installed properly. For more information, see Unable to install Rational® Functional TesterHCL OneTest™ UI Extension for Microsoft Visual Studio 2015, 2017 & 2019. **Note: Uninstalling Microsoft Visual Studio**: If you want to uninstall your existing Microsoft Visual Studio, you must first remove the Microsoft Visual Studio .NET Integration of Rational® Functional TesterHCL OneTest™ UI using the **Modify** option in Installation Manager.

Feature	Description
	You must then uninstall Microsoft Visual Studio.
Windows Desktop Application Testing (Next Generation)	The **Windows Desktop Application Testing (Next Generation)** option, which is a default selection, provides the necessary prerequisites to test the Windows desktop applications. The **Developer mode** is enabled automatically when you restart your computer, if it was not enabled earlier.
IBM® Rational® Test WorkbenchHCL OneTest™ Studio Extension for SAP Solutions	Supports extended functional testing of SAP GUI for Windows and SAP GUI for HTML. With the installation of this extension, you can record and play back scripts for the SAP UI with SAP controls and customized data verification of SAP controls.
Java 8 OpenJDK with Eclipse OpenJ9	Provides the Java 8.0 binaries required for Rational® Functional TesterHCL OneTest™ UI. **Notes:** ▪ If you want to shell share Rational® Functional TesterHCL OneTest™ UI along with other products that utilize OpenJDK, you must select **Java 8 OpenJDK with Eclipse OpenJ9** for only one of the shell shared products. You must not install multiple instances of OpenJDK in the same package group.
Jazz Eclipse Client for Rational Team Concert/Rational Quality Manager	Manages functional test assets using the Jazz source control management provided by the Jazz Eclipse Client. You must have a compatible version of Rational Team Concert or Rational Quality Manager server setup to use the Jazz source control manage-

Feature	Description
	ment. You must also have a platform that is also supported by the Rational Team Concert client.

b. **Optional:** To see the dependency relationships between features, select **Show Dependencies**.

c. **Optional:** Click a feature to view its brief description under **Details**.

d. Click **Next** to continue.

13. On the Summary page, review your choices before installing the Rational® Functional TesterHCL OneTest™ UI package. If you want to change the choices that you made on previous pages, click **Back** and make your changes. When you are satisfied with your installation choices, click **Install** to install the package.
    Result
    A progress indicator shows the percentage of the installation completed.
14. When the installation process is complete, a message confirms the success of the process.
    a. Click **View log file** to open the installation log file for the current session in a new window.
    b. Click **Finish**.

    After you install the product, the UI Test Agent runs automatically on your computer until you manually stop the UI Test Agent. Depending on the operating system of your computer, the UI Test Agent works as follows:
    - **Windows:** The UI Test Agent runs automatically and you can find the UI Test Agent in the system tray of your computer.
    - **Linux:** You must set the environment variable to run the UI Test Agent automatically. The UI Test Agent continues to run in the background. See the related information.
    - **macOS:** You must set the environment variable to run the UI Test Agent automatically. You can find the UI Test Agent on the toolbar of your computer. See the related information.

What to do next

Start the product as a root user or Sudo access for the first time. When you start the product, apply the product license key.

To start using the Visual Studio Integration feature, open the Visual Studio application and click **File > New > Functional Test Project**. The connector window opens within the Visual Studio and you can access the Rational® Functional TesterHCL OneTest™ UI environment.

To create a project, you must enable Java in the product.

Related information

Enabling Java environments on page 778

Modifying installations on page 228

Exporting functional test project items on page 863

Importing functional test project items on page 863

Installing and upgrading Rational® Functional TesterHCL OneTest™ UI in silent mode

In silent mode installation, you can install or upgrade Rational® Functional TesterHCL OneTest™ UI from the command line by using response files rather than using the Installation Manager graphical user interface.

Before you begin

You must have completed the following tasks:

- Uninstalled the existing version of the product before you install the latest version.
- Verified that your computer meets the system requirements.
- Ensured that your computer is connected to the internet.
- Downloaded the product bits.
- Downloaded and installed Installation Manager.

About this task

You can install Rational® Functional TesterHCL OneTest™ UI from the installation media or from a download package that is on the IBM® Passport Advantage® site from the HCL FlexNet Operations Portal.

To installor upgrade Rational® Functional TesterHCL OneTest™ UI in silent mode:

1. Create a response file that contains the commands and inputs for the installation process.
2. Install Rational® Functional TesterHCL OneTest™ UI by using the response file and Installation Manager installer:
   Choose from:
   - If you have already installed Installation Manager:
     a. Change to the following directory: `eclipse\tools`. For example: `cd C:\Program Files\IBM\Installation Manager\eclipse\tools.`
     b. Enter the installation command:
        - On Windows™ systems: `imcl.exe input response_file_path_and_name -log log_file_path_and_name -acceptLicense`
        - On Linux™ systems: `./imcl input response_file_path_and_name -log llog_file_path_and_name -acceptLicense`
   - If you have a complete Rational® Functional TesterHCL OneTest™ UI package that also contains Installation Manager, you can browse to the folder that contains the installation file of Installation Manager. You can also download Installation Manager that is required for installing Rational® Functional TesterHCL OneTest™ UI from the IBM website.

a. Change to the following directory: `InstallerImage_platform directory`.
b. Enter the installation command:
   - On Windows™ systems: `installc.exe input response_file_path_and_name -log log_file_path_and_name -acceptLicense`
   - On Linux™ systems: `./installc input response_file_path_and_name -log log_file_path_and_name -acceptLicense`

Hardware and Software requirements on page 201

Creating a response file manually on page 221

Working in silent mode with IBM Installation Manager

Creating a response file manually

To silently install or upgrade Rational® Functional TesterHCL OneTest™ UI, you can use a response file that contains the data required to install the product.

Before you begin

View the sample response file and know the parameters that you must change for your installation environment. For more information, see Sample response file. on page 221

For complete details about response files and silent installation, see the response file section in the Installation Manager information center.

1. Open the Sample response file topic to view the sample code. You can also find the sample response files in the *product install location*`\FunctionalTester\RQMAdapter\RTLM_BuildForgeScripts\SampleResponse`*product install location*`\hclonetest\RQMAdapter\RTLM_BuildForgeScripts\SampleResponse` location.
2. Copy the code from the sample response file to a text editor and save it as a `.xml` file.
3. Edit the required parameters that are specific to your installation environment and save the file.

What to do next

Installing and upgrading Rational Functional TesterHCL OneTest UI in silent mode on page 220

Sample response file

You can use the sample response file in this topic to install Rational® Functional TesterHCL OneTest™ UI. You must modify the parameter values for your installation environment. For instructions to create a response file, see Creating a response file manually on page 221.

Response file: A *response file* is an XML file that contains the data required to complete installation operations silently. Installation Manager uses response files to complete installation operations silently.

You can record a response file by recording preferences and installation actions in the graphical user interface of Installation Manager. Alternatively, you can create a response file manually by using the documented list of response file commands and preferences.

You can use one response file to install, update, or uninstall multiple products. For more information about response files, see the response file section in the Installation Manager information center.

Sample response file for installing Rational® Functional TesterHCL OneTest™ UI

```
<?xml version='1.0' encoding='UTF-8'?>
<agent-input>
<variables>
<variable name='sharedLocation' value='C:\Program Files\IBM\IBMIMShared'/>
</variables>
<server>
<repository location='C:/RFT/disk1/'/>
</server>
<profile id='IBM Software Delivery Platform' installLocation='C:\Program Files\IBM\SDP'>
<data key='cic.selector.arch' value='x86_64'/>
<data key='user.RPT_MX_VALUE' value='-Xmx4095m'/>
<data key='user.RPT_MX_VALUE_ORACLE' value='-Xmx4095m'/>
</profile>
<install>
<!-- IBM® Rational® Functional Tester 10.0.2 -->
<offering profile='IBM Software Delivery Platform' id='com.ibm.rational.functional.tester'
 version='10.0.200.FTO10_IBM-I20180910_1304'
 features='com.ibm.rpt.sdpcore,com.ibm.rpt.doc-isv,com.ibm.rtw.standard,com.ibm.rst.selenium,com.ibm.rft
.net2017,com.ibm.rst.rtc'/>
</install>
<preference name='com.ibm.cic.common.core.preferences.eclipseCache' value='${sharedLocation}'/>
<preference name='com.ibm.cic.agent.ui.displayInternalVersion' value='true'/>
</agent-input>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<agent-input>
<variables>
<variablename='sharedLocation'value='C:\Program Files\HCL\HCLIMShared'/>
</variables>
<server>
<repositorylocation='C:\HOTUI\Disk1'/>
</server>
<profileid='HCL OneTest'installLocation='C:\Program Files\HCL\HCLOneTest'>
<datakey='cic.selector.arch'value='x86_64'/>
<datakey='user.RPT_MX_VALUE'value='-Xmx4095m'/>
<datakey='user.RPT_MX_VALUE_ORACLE'value='-Xmx4095m'/>
</profile>
<install>
<!-- HCL OneTest™ UI 10.0.2 -->
<offeringprofile='HCL
 OneTest'id='com.hcl.onetest.ui'version='10.0.2.FTO10_HCL-I20190614_0929'features='com.ibm.rpt.sdpcore,c
om.ibm.rpt.doc-isv,com.ibm.rst.selenium,com.ibm.rtw.webgui,com.ibm.rtw.rft.integration,com.hcl.openj9.jd
k,com.ibm.rft.net2017,com.ibm.rpt.rit.integration'/>
</install>
<preferencename='com.ibm.cic.common.core.preferences.eclipseCache'value='${sharedLocation}'/>
<preferencename='com.ibm.cic.agent.ui.displayInternalVersion'value='true'/>
</agent-input>
```

Sample response file for upgrading Rational® Functional TesterHCL OneTest™ UI

```
<?xml version="1.0" encoding="UTF-8"?>
<!--The "acceptLicense" attribute has been deprecated. Use "-acceptLicense" command line option to
 accept license agreements.-->
<agent-input acceptLicense='true'>
<server>
<repository location='C:/RFT/disk1\'/>
</server>
<profile id='IBM Software Delivery Platform' installLocation='C:\Program Files\IBM\SDP'>
<data key='eclipseLocation' value='C:\Program Files\IBM\SDP'/>
<data key='user.import.profile' value='false'/>
<data key='cic.selector.os' value='win32'/>
<data key='cic.selector.ws' value='win32'/>
<data key='cic.selector.arch' value='x86'/>
<data key='user.help.option' value='remote'/>
<data key='user.help.url' value=''/>
<data key='cic.selector.nl' value='en'/>
</profile>
<<install modify='false'>
<offering id='com.ibm.rational.functional.tester' version='10.0.2.RFTO10-I20180708_1518' profile='IBM
 Software Delivery Platform' features='com.ibm.rft.agent,com.ibm.rft.java,com.ibm.rft.net2017'
 installFixes='none'/>
</install>
<preference name='com.ibm.cic.common.core.preferences.eclipseCache' value='C:\Program
 Files\IBM\IBMIMShared'/>
<preference name='com.ibm.cic.common.core.preferences.connectTimeout' value='30'/>
<preference name='com.ibm.cic.common.core.preferences.readTimeout' value='45'/>
<preference name='com.ibm.cic.common.core.preferences.downloadAutoRetryCount' value='0'/>
<preference name='offering.service.repositories.areUsed' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.ssl.nonsecureMode' value='false'/>
<preference name='com.ibm.cic.common.core.preferences.http.disablePreemptiveAuthentication'
 value='false'/>
<preference name='http.ntlm.auth.kind' value='NTLM'/>
<preference name='http.ntlm.auth.enableIntegrated.win32' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.keepFetchedFiles' value='false'/>
<preference name='PassportAdvantageIsEnabled' value='false'/>
<preference name='com.ibm.cic.common.core.preferences.searchForUpdates' value='false'/>
<preference name='com.ibm.cic.agent.ui.displayInternalVersion' value='false'/>
</agent-input>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--The "acceptLicense" attribute has been deprecated. Use "-acceptLicense" command line option to
 accept license agreements.-->
<agent-input acceptLicense='true'>
<server>
<repository location='C:\HOTUI\disk1\'/>
</server>
<profile id='HCL Products' installLocation='C:\Program Files\HCL\HCLOneTest'>
<data key='eclipseLocation' value='C:\Program Files\HCL\HCLOneTest'/>
<data key='user.import.profile' value='false'/>
<data key='cic.selector.os' value='win32'/>
<data key='cic.selector.ws' value='win32'/>
<data key='cic.selector.arch' value='x86'/>
<data key='user.help.option' value='remote'/>
<data key='user.help.url' value=''/>
<data key='cic.selector.nl' value='en'/>
</profile>
```

```
<<install modify='false'>
<offering id='HCL Products' id='com.hcl.onetest.ui' version=''10.0.2.FTO10_HCL-I20190614_092'
 profile='HCL Products' features='com.ibm.rft.agent,com.ibm.rft.java,com.ibm.rft.net2010'
 installFixes='none'/>
</install>
<preference name='com.ibm.cic.common.core.preferences.eclipseCache' value='C:\Program
 Files\HCL\HCLIMShared'/>
<preference name='com.ibm.cic.common.core.preferences.connectTimeout' value='30'/>
<preference name='com.ibm.cic.common.core.preferences.readTimeout' value='45'/>
<preference name='com.ibm.cic.common.core.preferences.downloadAutoRetryCount' value='0'/>
<preference name='offering.service.repositories.areUsed' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.ssl.nonsecureMode' value='false'/>
<preference name='com.ibm.cic.common.core.preferences.http.disablePreemptiveAuthentication'
 value='false'/>
<preference name='http.ntlm.auth.kind' value='NTLM'/>
<preference name='http.ntlm.auth.enableIntegrated.win32' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts' value='true'/>
<preference name='com.ibm.cic.common.core.preferences.keepFetchedFiles' value='false'/>
<preference name='PassportAdvantageIsEnabled' value='false'/>
<preference name='com.ibm.cic.common.core.preferences.searchForUpdates' value='false'/>
<preference name='com.ibm.cic.agent.ui.displayInternalVersion' value='false'/>
</agent-input>
```

Required parameters to modify

Based on your installation setup, you must edit parameters in your response file. The following tables show the parameters to modify.

Table 1. Parameters that must be modified based on your installation setup

Attributes	Description
acceptLicense	After you read and agree to the license terms, set the *acceptLicense* attribute value to `'true'`.
repository location	Specify the path of the Rational® Functional TesterHCL OneTest™ UI repository (disk1).
profile installLocation and *id*	Specify the Rational® Functional TesterHCL OneTest™ UI installation location.
features	Specify the features that you want to install with the Java scripting. Possible values: • `com.ibm.rft.net2015`: To install the Microsoft® Visual Studio .NET 2015 Integration feature of Rational® Functional TesterHCL OneTest™ UI. • `com.ibm.rft.net2017`: To install the Microsoft® Visual Studio .NET 2017 Integration feature of Rational® Functional TesterHCL OneTest™ UI. • `com.ibm.rft.net2019`: To install the Microsoft® Visual Studio .NET 2019 Integration feature of Rational® Functional TesterHCL OneTest™ UI.

Table 1. Parameters that must be modified based on your installation setup (continued)

Attributes	Description
	Note: You can only install one version of Microsoft Visual Studio Integration at a time. • `com.ibm.rft.rtc`: To install Jazz client in the same shell as Rational® Functional TesterHCL OneTest™ UI. **Note:** For information about the features and the prerequisites, see Rational Functional TesterHCL OneTest UI installation features on page 204.

Post-installation tasks

After you have installed your product package, complete the post-installation tasks or configure your product package as required.

Post-installation checklist

After you have installed your product, complete several tasks to configure and verify the installation.

About this task

Review the following information and ensure the post-installation steps are completed as required.

1. Verify your installation and ensure that you can start your product.
2. Configure your license for Rational® Functional TesterHCL OneTest™ UI.

Note: It is important to have a valid license configured for Rational® Functional TesterHCL OneTest™ UI in standalone as well as shell-shared environments. If a valid license is not available, the following functions are affected:

- Launching the Functional Test perspective
- Functionality related to the products that integrate with Rational® Functional TesterHCL OneTest™ UI
- Recording and playing back functional test scripts through the command line

Deploying the help content of Functional Tester in Visual Studio IDE

When you install Rational® Functional TesterHCL OneTest™ UI in Visual Studio IDE, the help content of Functional Tester also gets installed. You must then deploy this help content in Microsoft Help Viewer to access it from the Help menu of Visual Studio IDE.

Before you begin

You must have installed the following software:

- Rational® Functional TesterHCL OneTest™ UI in Visual Studio IDE. See Installing the product in Visual Studio IDE on page 237Installing Rational Functional TesterHCL OneTest UI on page 212.
- Microsoft Help Viewer. Refer to Microsoft Help Viewer Installation.

About this task

After you deploy the help content of Functional Tester in Visual Studio IDE, you can refer to the following content:

- Details about Rational® Functional TesterHCL OneTest™ UI proxy SDK
- Details about the product and functional testing
- Functional Tester API reference
- Functional Tester Proxy SDK API reference

1. Open Visual Studio.
2. Click **Help > Add and Remove Help Content**.
   The **Help Viewer** dialog is displayed.
3. Select the **Disk** option in the installation source.
4. Click to select the source location of the help content.
   The **Open** dialog is displayed.
5. Go to `<install_directory>\FunctionalTester\vsnet\MSDNHelp.`
   For example, the help content can be located in `C:\Program Files\HCL\HCLOneTest \FunctionalTester\vsnet\MSDNHelpC:\Program Files\IBM\FunctionalTester\vsnet \MSDNHelp`.
6. Select the `helpcontentsetup.msha` file, and then click **Open**.
   The the help content of Functional Tester is displayed.
7. Close the **Help Viewer** dialog.

Results

You have deployed the help content of Functional Tester to display in Visual Studio IDE.

What to do next

You can view the deployed help content by clicking **Help > View Help** in Visual Studio IDE.

Installation in the shell sharing mode

You can install Rational® Functional TesterHCL OneTest™ UI in the same package group with the other supported Eclipse-based products. These products share a common environment, or they are shell shared. Shell shared products have a common installation directory, `<install_directory>\IBM\SDP`. Rational® Functional TesterHCL OneTest™ UI installs as a perspective in the UI of the product with which it is shell-shared.

Prerequisites

You must be familiar with the installation and licensing of the following products that are compatible for installation in the shell shared mode with Rational® Functional TesterHCL OneTest™ UI:

Compatible product	Refer to
IBM® Engineering Workflow Management	Installing IBM® Engineering Workflow Management
IBM® Rational® Performance TesterHCL OneTest™ Performance	Installing IBM® Rational® Performance TesterHCL OneTest™ Performance Installing IBM® Rational® Performance TesterHCL OneTest™ Performance
IBM® Rational® Test WorkbenchHCL OneTest™ Studio	Installing IBM® Rational® Test WorkbenchHCL OneTest™ Studio
IBM® Rational® Team Concert™	Installing IBM® Rational® Team Concert™

Shell sharing with IBM® Engineering Workflow Management

You can install Rational® Functional TesterHCL OneTest™ UI in the shell sharing mode with Engineering Workflow Management Client Extension for Eclipse 4.x by using the following method:

1. Add the repositories of Rational® Functional TesterHCL OneTest™ UI and Engineering Workflow Management Client Extension in the IBM Installation Manager to install the products together in the shell sharing mode.
2. Select Rational® Functional TesterHCL OneTest™ UI and Engineering Workflow Management Client Extension to install the products together.

Shell sharing with Rational® Performance TesterHCL OneTest™ Performance

You can install Rational® Performance TesterHCL OneTest™ Performance with Rational® Functional TesterHCL OneTest™ UI in any sequence for the shell sharing mode. You can use one of the following methods:

If	Then
Rational® Functional TesterHCL OneTest™ UI is not installed	You must follow these steps: 1. Add the repositories of Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance in the IBM Installation Manager to install the products together in the shell sharing mode. 2. Select Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance to install the products together in a package group.
Rational® Functional TesterHCL OneTest™ UI is already installed	You must follow these steps:

If	Then
	1. Install Rational® Performance TesterHCL OneTest™ Performance and while installing Rational® Performance TesterHCL OneTest™ Performance, consider the following points: ◦ Choose the `IMShared` directory where Rational® Functional TesterHCL OneTest™ UI is already installed. ◦ Choose the `SDP` directory where Rational® Functional TesterHCL OneTest™ UI is already installed.

Shell sharing with Rational® Test WorkbenchHCL OneTest™ Studio

You can install Rational® Functional TesterHCL OneTest™ UI in the shell sharing mode with the other packages of Rational® Test WorkbenchHCL OneTest™ Studio by using the following method:

- In the package group of Rational® Test WorkbenchHCL OneTest™ Studio, you can install Rational® Functional TesterHCL OneTest™ UI with the other packages simultaneously.

Shell sharing with IBM® Rational® Team Concert™

You can install Rational® Functional TesterHCL OneTest™ UI in the shell sharing mode with Rational® Team Concert™ 6.06 Client Extension for Eclipse by using the following method:

If	Then
Rational® Functional TesterHCL OneTest™ UI is not installed	You must follow these steps: 1. Add the repositories of Rational® Functional TesterHCL OneTest™ UI and Rational® Team Concert™ Client Extension in the IBM Installation Manager to install the products together in the shell sharing mode. 2. Select Rational® Functional TesterHCL OneTest™ UI and Rational® Team Concert™ Client Extension to install the products together.
Rational® Functional TesterHCL OneTest™ UI is already installed	Install Rational® Team Concert™ Client Extension and while installing it, choose the `SDP` directory where Rational® Functional TesterHCL OneTest™ UI is already installed.

Modifying installations

The Modify Packages wizard in the IBM® Installation Manager enables you to change the language and feature selections of an installed product package. You can also use the Modify Packages wizard to install new features that might be included in a package update, such as a refresh pack.

Before you begin

By default, Internet access is required unless the repository preferences points to a local update site. See the Installation Manager help for more information.

Note:

- Before you modify Rational® Functional TesterHCL OneTest™ UI, close the Eclipse and Visual Studio IDEs, as well as any open web browsers, and all other applications that are enabled by Rational® Functional TesterHCL OneTest™ UI.
- Before you modify Rational® Functional TesterHCL OneTest™ UI, close all programs that were installed using Installation Manager.

1. From the Start page of the Installation Manager, click the **Modify** icon.
2. In the Modify Packages wizard, select the installation location for the Rational® Functional TesterHCL OneTest™ UI product package and click **Next**.
3. On the Modify page, under Languages, select the languages for the package group, then click **Next**.
   The corresponding national language translations for the user interface and documentation for the packages will be installed. Note that your choices apply to all packages installed under this package group.
4. On the Features page, select the package features that you want to install or remove.
   a. To learn more about a feature, click the feature and review the brief description under **Details**.
   b. If you want to see the dependency relationships between features, select **Show Dependencies**. When you click a feature, any features that depend on it and any features that are its dependents are shown in the Dependencies window. As you select or exclude features in the packages, Installation Manager will automatically enforce any dependencies with other features and display updated download size and disk space requirements for the installation.
5. When you are finished selecting features, click **Next**.
6. On the Summary page, review your choices before modifying the installation package, and then click **Modify**.
7. **Optional:** When the modification process completes, click **View Log File** to see the complete log.

Uninstalling Rational® Functional TesterHCL OneTest™ UI

The Uninstall Packages option in the Installation Manager enables you to uninstall packages from a single installation location. You can also uninstall all the installed packages from every installation location.

Before you begin

You must have completed the following tasks before you uninstall Rational® Functional TesterHCL OneTest™ UI:

- Logged in to the system using the same user account that you used to install the product packages.
- Closed the Eclipse and Visual Studio IDEs, as well as any open web browsers, and all other applications that are enabled by Rational® Functional TesterHCL OneTest™ UI.

To uninstall Rational® Functional TesterHCL OneTest™ UI using Installation Manager

1. Close the programs that you installed using Installation Manager.
2. On the Start page click **Uninstall Packages**.
3. In the Uninstall Packages page, select the Rational® Functional TesterHCL OneTest™ UI product package that you want to uninstall. Click **Next**.
4. In the Summary page, review the list of packages that will be uninstalled and then click **Uninstall**.
   Result
   The Complete page is displayed after the uninstallation finishes.
5. Click **Finish** to exit the wizard.

Installation of the product by using the stand-alone installer

This guide intended for an administrator and describes how to install the product by using stand-alone installer on different operating systems.

You can install the product by using any of the following methods:

- GUI mode
- Console mode
- Silent mode

Installation of the product on Windows systems

You can find information about installing the product software on Windows system by using the stand-alone installer.

You can install the product by using any of the following methods:

- GUI mode
- Silent mode

Installing the product in the GUI mode on Windows systems

You can get started with the desktop client on Windows system after installing the product in the GUI mode by using the stand-alone installer.

Before you begin

- You must be an administrator.

  Note: If you are a non-admin user, you must have the credentials of an administrator to start the installation by using the **Run as Administrator** option.

- You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling the product in the GUI mode on Windows systems on page 284.
- Verified the software and hardware requirements. See System Requirements on page 13.
- Cleared files from the `Temp` directory, restarted the computer, and started stand-alone installer as an Administrator.

About this task

The 32-bit stand-alone installer of Rational® Functional TesterHCL OneTest™ UI supports only SAP extension. The test extensions such as SAP, Citrix, and Siebel are supported only for 32-bit stand-alone Rational® Performance TesterHCL OneTest™ Performance installer and 32-bit shell-shared installer that are installed on Windows system 64-bit operating system. You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Right-click the product installer file and select **Run as Administrator**.

   The GUI window is displayed.

3. Complete the following steps:

Step	Action
Language selection	Select the language from the drop-down list to view the installation instructions and the product in your preferred language. The drop-down list displays the languages based on the display language that is set on your computer.
Introduction	Read through the details on the window, and then click **Next**.
License Agreement	Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
Microsoft License Terms	Read the Microsoft Software license agreement carefully, and then select the **I accept** check box to install the execution agent that provides you with the necessary prerequisites to test the hybrid and native mobile applications, and Windows desktop applications. Click **Next** to continue. **Note:** After the installation, the execution agent starts automatically whenever you start Rational® Functional TesterHCL OneTest™ UI.
Destination Folder	Browse for the location or directory where you want to install the product, and then click **Next**. **Note:** You must select any other directory if the default directory is not empty. The default locations for 32-bit and 64-bit installers are as follows:

Step	Action
	◦ For 32-bit: `C:\Program Files(x86)\HCL` ◦ For 64-bit: `C:\Program Files\HCL`
Visual Studio Integration	Select the version of the Visual Studio with which you want to integrate the product, and then click **Next**.
Windows Application Testing	Read the information about the Developer Mode activation, and then click **Next**.
Installation Summary	Read through the installation details, and then click **Install**.
Installing...	Wait for the installation to complete.
Install Complete	Click **Done** after the installation of the product is complete.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Creating a properties file on Windows systems

To install the product in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 233 through 4 on page 234.
- Copying the contents of the sample properties file to your local properties file. See step 5 on page 234.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Right-click the **Command Prompt** application and select **Run as Administrator**.

3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd C:\users\Downloads
4. Run the following command to generate the properties file:

   ```
   <installer_file.exe> -r <location of the installer.properties file>
   ```

   For example, HCL-OneTest-Performance-Windows-x64-v10.1.0.0.exe -r C:\users\Downloads \installer.propertiesHCL-OneTest-UI-Windows-x64-v10.1.0.0.exe -r C:\users\Downloads\installer.properties
5. Create your properties file by copying the content of the following sample properties file:

   ```
   #Host Name
   # Replay feature output
   # ---------------------
   # This file was built by the Replay feature of Installer.
   # It contains variables that were set by Panels, Consoles or Custom Code.



   #Destination Folder
   #------------------
   USER_INSTALL_DIR=C:\\Program Files\\HCL
   ```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing the product in the silent mode on Windows systems

To get started with the desktop client, you can install the product on Windows system by using the silent mode installation method.

Before you begin

- You must be an administrator.

  Note: If you are a non-admin user, you must have the credentials of an administrator to start the installation by using the **Run as Administrator** option.

- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling the product in the silent mode on Windows systems on page 285.
  - Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.

- Created a properties file that contains the inputs for the installer during the installation process. See Creating a properties file on Windows systems on page 233.
- Verified the software and hardware requirements. See System Requirements on page 13.
- Cleared files from the `Temp` directory, restarted the computer, and started stand-alone installer as an Administrator.

About this task

The 32-bit stand-alone installer of Rational® Functional TesterHCL OneTest™ UI supports only the SAP extension. The SAP, Citrix, and Siebel tests are supported only for 32-bit stand-alone Rational® Performance TesterHCL OneTest™ Performance installer and 32-bit shell-shared installer that are installed on Windows system 64-bit operating system. You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Right-click the **Command Prompt** application and select **Run as Administrator**.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd C:\users\Downloads
4. Extract the downloaded installer file.
5. Run the following command to install the product:

   ```
   <installer_name> -i silent -l <language_code> -f <location_of_the_installer.properties_file_name>
   ```

   For example, you can run the following command to install Rational® Performance TesterHCL OneTest™ PerformanceRational® Functional TesterHCL OneTest™ UI 10.2.0 in French:

HCL-OneTest-Performance-Windows-x64-v10.2.0.0.exe -i silent -l fr -f C:\users\Downloads \installer.propertiesHCL-OneTest-UI-Windows-x64-v10.2.0.0.exe -i silent -l fr -f C:\users\Downloads \installer.properties

The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-l** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` file to know more about the problem that occurred during the installation process. The error log is located at `\<installation directory>\Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `\<installation directory>\Logs`.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Installing the product in Visual Studio IDE

To integrate Rational® Functional TesterHCL OneTest™ UI with the Microsoft Visual Studio and start using Rational® Functional TesterHCL OneTest™ UI in the Visual Studio environment, you must first install Rational® Functional TesterHCL OneTest™ UI by using the stand-alone installer.

Before you begin

- You must be an administrator.

  Note: If you are a non-admin user, you must have the credentials of an administrator to start the installation by using the **Run as Administrator** option.

- You must have installed the appropriate version of Visual Studio with which you want to integrate Rational® Functional TesterHCL OneTest™ UI.
- Verified the software and hardware requirements. See System Requirements on page 13.

  Note: Integration of Rational® Functional TesterHCL OneTest™ UI with Microsoft Visual Studio might not be successful if the mandatory prerequisite software and hardware are not installed, or if Microsoft Visual Studio is not installed properly. For more information, refer to Unable to install HCL OneTest™ UI Extension for Microsoft Visual Studio 2015, 2017 & 2019

About this task

You must install the 64-bit installer on the Windows system 64-bit operating system. You cannot update, modify, or roll back the product by using the stand-alone installer.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Right-click the *.exe* file and select **Run as Administrator**.

   The GUI window is launched.
3. On the **Introduction** window, read through the details, and click **Next**.
4. Read the license agreement carefully, select the **I accept** check box and then click **Next**.
5. Browse the location or directory where you want to install the product and click **Next**.

   Note: You must select a different directory if the default directory is not empty. The default location for the 64-bit installer is `C:\Program Files\HCL`.

6. Select the version of the Visual Studio with which you want to integrate the product, and then click **Next**.
7. Read through the installation details and click **Install**.
8. Click **Done** after the installation is complete.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

To start using the Visual Studio Integration feature, open the Visual Studio application and click **File > New > Functional Test Project**. The connector window opens within the Visual Studio and you can access the Rational® Functional TesterHCL OneTest™ UI environment.

Deploying the help content of Functional Tester in Visual Studio IDE

When you install Rational® Functional TesterHCL OneTest™ UI in Visual Studio IDE, the help content of Functional Tester also gets installed. You must then deploy this help content in Microsoft Help Viewer to access it from the Help menu of Visual Studio IDE.

Before you begin

You must have installed the following software:

- Rational® Functional TesterHCL OneTest™ UI in Visual Studio IDE. See Installing the product in Visual Studio IDE on page 237Installing Rational Functional TesterHCL OneTest UI on page 212.
- Microsoft Help Viewer. Refer to Microsoft Help Viewer Installation.

About this task

After you deploy the help content of Functional Tester in Visual Studio IDE, you can refer to the following content:

- Details about Rational® Functional TesterHCL OneTest™ UI proxy SDK
- Details about the product and functional testing
- Functional Tester API reference
- Functional Tester Proxy SDK API reference

1. Open Visual Studio.
2. Click **Help > Add and Remove Help Content**.
   The **Help Viewer** dialog is displayed.
3. Select the **Disk** option in the installation source.
4. Click to select the source location of the help content.
   The **Open** dialog is displayed.
5. Go to `<install_directory>\FunctionalTester\vsnet\MSDNHelp`.
   For example, the help content can be located in `C:\Program Files\HCL\HCLOneTest \FunctionalTester\vsnet\MSDNHelpC:\Program Files\IBM\FunctionalTester\vsnet \MSDNHelp`.
6. Select the `helpcontentsetup.msha` file, and then click **Open**.

The the help content of Functional Tester is displayed.

7. Close the **Help Viewer** dialog.

Results

You have deployed the help content of Functional Tester to display in Visual Studio IDE.

What to do next

You can view the deployed help content by clicking **Help > View Help** in Visual Studio IDE.

Changing the language of the product on Windows systems

When you want to use the product in a different language other than the display language of your computer, then you can change the language of the product by using the command-line interface.

About this task

Important: After you changed the language and closed the application for any reason, the product displays in the language that is set in your computer if you re-open the product.

For example, consider the display language of your computer is set as French and you change the language of the product to Japanese by using the command-line interface. When you close the product and open it again, the product displays in the French language.

1. Right-click the **Command Prompt** application and select **Run as Administrator**.
2. Run the following command to change the directory to a location where you installed the product:

   ```
   cd <directory path>
   ```

   For example, cd C:\Program Files\HCL\HCLOneTest

3. Run the following command to change the language of the product:

   ```
   eclipse.exe -nl <language_code>
   ```

   Where *<language_code>* is the code of your preferred language.

   The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

For example, eclipse.exe -nl ja

Result

The product opens in the Japanese language.

Results

You have changed the language of the product until you restart the product.

Installation of the product software on Linux

You can find information about installing the product software on Linux by using the stand-alone installer.

You can install the product software by using any of the following methods:

- GUI mode
- Console mode
- Silent mode

Installing the product in the GUI mode on Linux

To get started with the desktop client on Linux, you can install the product in the GUI mode by using the stand-alone installer.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling the product in the GUI mode on Linux on page 287.
- Installed Konsole on your computer. The Konsole application is required to enable the shortcut icon of the product after installation. You can install Konsole by running the apt-get install konsole command in the terminal.

  Note: You need not install Konsole on the computer that is running RHEL 8 or later.

- Verified that the following libraries are available on your computer, if you want to install Rational® Functional TesterHCL OneTest™ UI:
  - libXp.so.6
  - libXm.so.4

Note: For more information on the library dependencies on Linux, refer to Library dependency on Linux.

- Created a soft link to libnsl.so.2 by running the following command:

```
sudo ln -s /usr/lib64/libnsl.so.2 /usr/lib64/libnsl.so
```

- Verified the software and hardware requirements. See System Requirements on page 13.

About this task

You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by

using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as a root user.
3. Run the following command to extract the zip file:

   ```
   unzip <filename>
   ```

   Result

   The GUI window is displayed.
4. Select the language from the drop-down list to view the installation instructions and the product in your preferred language.

   The drop-down list displays the languages based on the display language that is set on your computer.
5. Read through the details on the **Introduction** window, and then click **Next**.
6. Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
7. Browse for the location or directory where you want to install the product, and then click **Next**.

   Note: The default installation directory is `/opt/HCL`.
8. Read through the installation details and, then click **Install**.
9. Click **Done** after the installation of the product is complete.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Installing the product in the console mode on Linux

To install the product on Ubuntu and Red Hat Enterprise Linux (RHEL) operating systems where GUI mode is not supported, you can use the console-mode installation method.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling the product in the console mode on Linux on page 287.
- Installed Konsole on your computer. The Konsole application is required to enable the shortcut icon of the product after installation. You can install Konsole by running the apt-get install konsole command in the terminal.

Note: You need not install Konsole on the computer that is running RHEL 8 or later.

- Verified that the following libraries are available on your computer, if you want to install Rational® Functional TesterHCL OneTest™ UI:
  - libXp.so.6
  - libXm.so.4

Note: For more information on the library dependencies on Linux, refer to Library dependency on Linux.

- Verified the software and hardware requirements. See System Requirements on page 13.

About this task

In console mode, the installation process is an interactive and text-based. You can install the product in the console mode only on the Ubuntu and RHEL operating systems.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the appropriate number during the installation of the product.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as a root user.
3. Run the following command to change the directory from root to the directory where you downloaded the installer zip file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/

4. Run the following command to extract the downloaded zip file:

   ```
   unzip <product package name>
   ```

   For example, unzip HCL-OneTest-UI-Linux-x64-v10.1.0.0.zipunzip HCL-OneTest-Performance-Linux-x64-v10.1.0.0.zip

5. Run the following command to begin the installation process:

   ```
   ./<installer_name.bin> -i console
   ```

   For example, ./HCL-OneTest-UI-Linux-x64-v10.1.0.0.bin -i console./HCL-OneTest-Performance-Linux-x64-v10.1.0.0.bin -i console

   The command-line interface displays the list of preferred languages based on the display language that is set on your computer.

   Note: To identify the display language of your computer, the command-line interface displays an arrow mark (->) that precedes the name of the language.

6. Enter a number to select your preferred language, and then **Enter**.
   Result
   The installation instructions are displayed in your preferred language.
7. Review the license agreement and perform the following actions:
   - Read the license agreement by pressing any key along with the **Enter** key to continue reading the license agreement.
   - At any time, press `0 (zero)` along with the **Enter** key to go to the end of the license agreement.
   - After you read the license agreement, enter `Y` to accept the license agreement and continue with the installation.

Note: You can type `N` if you want to cancel the installation.

8. Provide the path where you want to install the product and perform the following steps:
   a. Press **Enter** to install the product in the default path: `/opt/HCL`.
   b. Enter the complete path of the directory and press **Enter** to install the product in the new directory path.

Note: You can type `Back` and press **Enter** to go to the previous step if you want to modify your inputs during the installation process.

9. Review the pre-installation summary, and then press **Enter** to continue with the installation process.
10. Press **Enter** to exit the installer.

Result

Note: If the installation of the product is not successful, you can check the errors that occurred during the installation in the log file. The log file is available at the location: `/opt/HCL/Logs`.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Creating a properties file on Linux

To install the product in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 245 through 4 on page 246 .
- Copying the contents of the sample properties file to your local properties file. See 5 on page 246.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by

using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as a root user.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/

4. Run the following command to generate the properties file:

   ```
   <installer_file.bin> -r <location of the installer.properties file>
   ```

   For example,

   For example, HCL-OneTest-Performance-Linux-x64-v10.1.0.0.bin -r /root/Downloads/installer.propertiesHCL-OneTest-UI-Linux-x64-v10.1.0.0.bin -r /root/Downloads/installer.properties

5. Create your properties file by copying the content of the following sample properties file:

   ```
   #Host Name
   # Replay feature output
   # ---------------------
   # This file was built by the Replay feature of Installer.
   # It contains variables that were set by Panels, Consoles or Custom Code.



   #Destination Folder
   #------------------
   USER_INSTALL_DIR=/opt/HCL
   ```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing the product in the silent mode on Linux

To get started with the product, you can install the product on Linux by using the silent mode installation method.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling the product in the silent mode on Linux on page 288.
- Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.

- Created a properties file that contains the inputs for the installer during the installation process. See Creating a properties file on Linux on page 245.
- Verified the software and hardware requirements. See System Requirements on page 13.
- Verified that the following libraries are available on your computer, if you want to install Rational® Functional TesterHCL OneTest™ UI:
  - libXp.so.6
  - libXm.so.4

Note: For more information on the library dependencies on Linux, refer to Library dependency on Linux.

About this task

You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as a root user.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/

4. Run the following command to extract the downloaded zip file:

   ```
   unzip <product package name>
   ```

   For example, unzip HCL-OneTest-UI-Linux-x64-v10.1.0.0.zipunzip HCL-OneTest-Performance-Linux-x64-v10.1.0.0.zip

5. Run the following command to install the product:

```
<installer_name> -i silent -l <language_code> -f <location_of_the_installer.properties_file_name>
```

For example, you can run the following command to install Rational® Performance TesterHCL OneTest™ PerformanceRational® Functional TesterHCL OneTest™ UI 10.2.0 in French:

HCL-OneTest-Performance-Linux-x64-v10.2.0.0.bin -i silent -l fr -f /root/Downloads/installer.propertiesHCL-OneTest-UI-Linux-x64-v10.2.0.0.bin -i silent -l fr -f /root/Downloads/installer.properties

The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-l** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` to know more about the problem that occurred during the installation process. The error log is located at `/<installation directory>/Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `/<installation directory>/Logs`.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Increasing the number of file handles on Linux™ workstations

For best product performance, increase the number of file handles above the default setting of 1024 handles.

About this task

Important: Before you work with your product, increase the number of file handles. Most of the products use more than the default limit of 1024 file handles per process. A system administrator might need to make this change.

Exercise caution when using the following steps to increase your file descriptors on Linux™. If the instructions are not followed correctly, the computer might not start correctly.

1. Log in as root.

   If you do not have root access, you must obtain it before continuing.

2. Change to the `etc` directory.

   Attention: If you decide to increase the number of file handles in the next step, *do not* leave an empty `initscript` file on your computer. If you do so, your computer will not start up the next time that you turn it on or restart.

3. Use the vi editor to edit the `initscript` file in the `etc` directory. If this file does not exist, type `vi initscript` to create it.
4. On the first line, type `ulimit -n 30000`.
   The point is that 30000 is significantly larger than 1024, the default value on most Linux™ computers.

   Important: Do not set the number of handles too high, because doing so can negatively impact system-wide performance.

5. On the second line, type `eval exec "$4"`.
6. Save and close the file after making sure that you have completed steps 4 and 5.

   Note: Ensure that you follow the steps correctly. If this procedure is not completed correctly, your computer will not start.

7. **Optional:** Restrict the number of handles available to users or groups by modifying the `limits.conf` file in the `etc/security` directory.
   If you do not have this file, consider using a smaller number in step 4 in the previous procedure (for example, 2048). Do this so that most users have a reasonably low limit on the number of open files that are allowed per process. If you use a relatively low number in step 4, it is less important to do this. However, if you set a high number in step 4 earlier and you do not establish limits in the `limits.conf` file, computer performance can be significantly reduced.

The following sample `limits.conf` file restricts all users, and then sets different limits for others afterwards. This sample assumes that you set handles to 8192 in step 4 earlier.

```
*      soft nofile 1024
*      hard nofile 2048
root   soft nofile 4096
root   hard nofile 8192
user1  soft nofile 2048
user1  hard nofile 2048
```

Note that the `*` in the preceding example sets the limits for all users first. These limits are lower than the limits that follow. The root user has a higher number of allowable handles open, while the number that is available to user1 is between the two. Make sure that you read and understand the documentation that the `limits.conf` file contains before making changes.

What to do next

For more information on the ulimit command, see the main page for ulimit in the Linux™ documentation.

Changing the language of the product on Linux

When you want to use the product in a different language other than the display language of your computer, then you can change the language of the product by using the command-line interface.

About this task

Important: After you changed the language and closed the application for any reason, the product displays in the language that is set in your computer if you re-open the product.

For example, consider the display language of your computer is set as French and you change the language of the product to Japanese by using the command-line interface. When you close the product and open it again, the product displays in the French language.

1. Open the terminal and log in as a root user.
2. Run the following command to change the directory to a location where you installed the product:

   ```
   cd <directory path>
   ```

3. Run the following command to change the language of the product:

   ```
   ./eclipse -nl <language_code>
   ```

   Where *<language_code>* is the code of your preferred language.

   The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

For example, ./eclipse -nl ja

Result

The product opens in the Japanese language.

Results

You have changed the language of the product until you restart the product.

Installation of IBM® Rational® Performance TesterHCL OneTest™ Performance Agent by using the stand-alone installer

To get started with IBM® Rational® Performance TesterHCL OneTest™ Performance Agent, you must install the product on Windows system or Linux or macOS by using the stand-alone installer.

You can install the product by using any of the following methods:

- GUI mode
- Console mode
- Silent mode

Installation of the Agent on Windows systems

You can find information about installing Rational® Performance TesterHCL OneTest™ Performance Agent on Windows system by using the stand-alone installer.

You can install the Agent by using any of the following methods:

- GUI mode
- Silent mode

Installing Agent in the GUI mode on Windows systems

You can install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on different computers to apply load on the server that hosts the application under test or to run multiple Web UI tests. When you want to run the test scripts on any computer, you must install Rational® Performance TesterHCL OneTest™ Performance Agent.

Before you begin

- You must be an administrator.

  Note: If you are a non-admin user, you must have the credentials of an administrator to start the installation by using the **Run as Administrator** option.

- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling Agent in the GUI mode on Windows systems on page 285.
  - Verified the software and hardware requirements. See System Requirements on page 13.
  - Cleared files from the `Temp` directory, restarted the computer, and started stand-alone installer as an Administrator.

About this task

All 32-bit test extensions such as SAP, Citrix, and Siebel are supported only for 32-bit Rational® Performance TesterHCL OneTest™ Performance Agent. The product installer does not support the following features:

- Updates, modifications, or rollback
- Shell sharing in Eclipse
- Sharing of common components
- Integration with Visual Studio for Rational® Functional TesterHCL OneTest™ UI

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

If you do not want to configure the agent during the installation of the product, then you can leave the configuration fields blank. Later, when you decide to configure the agent, you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`\HCL\HCLoneTest\Majordomo.config`

Note: The **URL Alias** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Right-click the product installer file and select **Run as Administrator**.

   The GUI window is displayed.
3. Select the language from the drop-down list to view the installation instructions and the product in your preferred language.

   The drop-down list displays the languages based on the display language that is set on your computer.
4. Read through the details on the **Introduction** window, and then click **Next**.
5. Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
6. Read the Microsoft Software license agreement carefully, and then select the **I accept** check box to install the execution agent that provides you with the necessary prerequisites to test the hybrid and native mobile applications, and Windows desktop applications. Click **Next** to continue.

Note: After the installation, the execution agent starts automatically when you run AFT test suites and it stops after the playback is completed.

7. Browse for the location or directory where you want to install the product, and then click **Next**.

Note: You must select any other directory if the default directory is not empty. The default locations for 32-bit and 64-bit installers are as follows:

- For 32-bit: `C:\Program Files(x86)\HCL`
- For 64-bit: `C:\Program Files\HCL`

8. **Optional:** Perform the following steps to configure the agent:

a. Specify the values for the following parameters for Rational® Functional TesterHCL OneTest™ UI:

Field name	Description	Example
Host name	The hostname of Rational® Functional TesterHCL OneTest™ UI.	localhost
Port	The port number of Rational® Functional TesterHCL OneTest™ UI.	7080

b. Specify the values for the following parameters for Rational® Test Automation ServerHCL OneTest™ Server:

Field name	Description	Example
Host name	The hostname of Rational® Test Automation ServerHCL OneTest™ Server. **Note:** ▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server must be resolvable through a Domain Name Server (DNS). ▪ An IP address cannot be the hostname of Rational® Test Automation ServerHCL OneTest™ Server.	localhost

Field name	Description	Example
	 ▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server through host files must not be specified. For example, `/etc/hosts` or `C:/Windows/system32/drives/etc/host`	
Port	The port number of Rational® Test Automation ServerHCL OneTest™ Server.	443
Token	An offline user token that is created from Rational® Test Automation ServerHCL OneTest™ Server.	eyJhbGciOiJIUzI1NiIsInR
URL Alias	The name of the **URL Alias** that you provided during the creation of the team space in Rational® Test Automation ServerHCL OneTest™ Server.	testteam

9. Read through the installation details and, then click **Install**.
10. Click **Done** after the installation of the product is complete.

Results

You have installed the agent on your computer.

Creating the properties file on Windows systems for Rational® Performance TesterHCL OneTest™ Performance Agent

To install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 255 through 4 on page 256.
- Copying the contents of the sample properties file to your local properties file. See step 5 on page 256.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Right-click the **Command Prompt** application and select **Run as Administrator**.
3. Run the following command to change the directory to a location where you downloaded the installer file:

```
cd <directory path>
```

For example, cd C:\users\Downloads

4. Run the following command to generate the properties file:

```
<installer_file.exe> -r <location of the installer.properties file>
```

For example, HCL-OneTest-Performance-Agent-Windows-x64-v10.1.0.0.exe -r C:\users\Downloads \installer.properties

5. Create your properties file by copying the content of the following sample properties file:

```
# Replay feature output
# ---------------------
# This file was built by the Replay feature of Installer.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Destination Folder
#------------------
USER_INSTALL_DIR=C:\\Program Files\\HCL

#HCL Load Generation Agent Configuration
#---------------------------------------
WORKBENCH_HOSTNAME=<host name of HCL OneTest UI>
WORKBENCH_PORT=7080
SERVER_HOSTNAME=<host name of server>
SERVER_PORT=6969
SERVER_TOKEN=<server token>
SERVER_URL_ALIAS=<name of the URL alias for the team space>
```

```
# Replay feature output
# ---------------------
# This file was built by the Replay feature of Installer.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Destination Folder
#------------------
USER_INSTALL_DIR=C:\\Program Files\\HCL

#HCL Load Generation Agent Configuration
#---------------------------------------
WORKBENCH_HOSTNAME=<host name of HCL OneTest Performance>
```

```
WORKBENCH_PORT=7080
SERVER_HOSTNAME=<host name of server>
SERVER_PORT=6969
SERVER_TOKEN=<server token>
SERVER_URL_ALIAS=<name of the URL alias for the team space>
```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing Agent in the silent mode on Windows systems

You can install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on different computers to apply load on the server that hosts the application under test or to run multiple Web UI tests. When you want to run the test scripts on any computer, you must install Rational® Performance TesterHCL OneTest™ Performance Agent.

Before you begin

- You must be an administrator.

  Note: If you are a non-admin user, you must have the credentials of an administrator to start the installation by using the **Run as Administrator** option.

- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling Agent in the silent mode on Windows systems on page 286.
  - Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.
  - Created a properties file that contains the inputs for the installer during the installation process. See Creating the properties file on Windows systems for Rational Performance TesterHCL OneTest Performance Agent on page 255.
  - Verified the software and hardware requirements. See System Requirements on page 13.
  - Cleared files from the `Temp` directory, restarted the computer, and started stand-alone installer as an Administrator.

About this task

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

If you decided to configure the agent, later, then you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`\HCL\HCLoneTest\Majordomo.config`

Note: The **SERVER_URL_ALIAS** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Right-click the **Command Prompt** application and select **Run as Administrator**.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd C:\users\Downloads
4. Extract the downloaded product installer file.
5. Run the following command to install the product:

   ```
   <installer_name> -i silent -l <language_code> -f <location_of_the_installer.properties_file_name>
   ```

   For example, you can run the following command to install Rational® Performance TesterHCL OneTest™ Performance Agent 10.2.0 in French:

   HCL-OneTest-Performance-Agent-Windows-x64-v10.2.0.0.exe -i silent -l fr -f C:\users\Downloads \installer.properties

   The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-l** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` file to know more about the problem that occurred during the installation process. The error log is located at `\<installation directory>\Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `\<installation directory>\Logs`.

Results

You have installed the product on your computer.

Installation of the Agent on Linux

You can find information about installing Rational® Performance TesterHCL OneTest™ Performance Agent on Linux by using the stand-alone installer.

You can install the Agent by using any of the following methods:

- GUI mode
- Console mode
- Silent mode

Installing Agent in the GUI mode on Linux

You can install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on different computers to apply load on the server that hosts the application under test or to run multiple Web UI tests. When you want to run the test scripts on any computer, you must install Rational® Performance TesterHCL OneTest™ Performance Agent.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling Agent in the GUI mode on Linux on page 289.
- Verified the software and hardware requirements. See System Requirements on page 13.

About this task

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

If you do not want to configure the agent during the installation of the product, then you can leave the configuration fields blank. Later, when you decide to configure the agent, you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`/HCL/HCLoneTest/Majordomo.config`

Note: The **URL Alias** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Open the terminal and log in as a root user.
3. Run the following command to extract the zip file:

   ```
   unzip <filename>
   ```

   Result

The GUI window is displayed.

4. Select the language from the drop-down list to view the installation instructions and the product in your preferred language.

   The drop-down list displays the languages based on the display language that is set on your computer.

5. Read through the details on the **Introduction** window, and then click **Next**.
6. Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
7. Browse for the location or directory where you want to install the product, and then click **Next**.

   Note: The default installation directory is `/opt/HCL`.

8. **Optional:** Perform the following steps to configure the agent:

   a. Specify the values for the following parameters for Rational® Functional TesterHCL OneTest™ UI:

Field name	Description	Example
Host name	The hostname of Rational® Functional TesterHCL OneTest™ UI.	localhost
Port	The port number of Rational® Functional TesterHCL OneTest™ UI.	7080

   b. Specify the values for the following parameters for Rational® Test Automation ServerHCL OneTest™ Server:

Field name	Description	Example
Host name	The hostname of Rational® Test Automation ServerHCL OneTest™ Server. **Note:** ▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server must be resolvable through a Domain Name Server (DNS). ▪ An IP address cannot be the hostname of Rational® Test Automation ServerHCL OneTest™ Server.	localhost

Field name	Description	Example
	▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server through host files must not be specified. For example, `/etc/hosts` or `C:/Windows/system32/drives/etc/host`	
Port	The port number of Rational® Test Automation ServerHCL OneTest™ Server.	443
Token	An offline user token that is created from Rational® Test Automation ServerHCL OneTest™ Server.	eyJhbGciOiJIUzI1NiIsInR
URL Alias	The name of the **URL Alias** that you provided during the creation of the team space in Rational® Test Automation ServerHCL OneTest™ Server.	testteam

9. Read through the installation details and, then click **Install**.
10. Click **Done** after the installation of the product is complete.

Results

You have installed the agent on your computer.

Installing Agent in the console mode on Linux

To install the IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on Ubuntu and Red Hat Enterprise Linux (RHEL) operating systems where GUI mode is not supported, you can use the console-mode installation method.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling the product in the console mode on Linux on page 287.
- Installed Konsole on your computer. The Konsole application is required to enable the shortcut icon of the product after installation. You can install Konsole by running the apt-get install konsole command in the terminal.

Note: You need not install Konsole on the computer that is running RHEL 8 or later.

- Verified that the following libraries are available on your computer, if you want to install Rational® Functional TesterHCL OneTest™ UI:
  - libXp.so.6
  - libXm.so.4

Note: For more information on the library dependencies on Linux, refer to Library dependency on Linux.

- Verified the software and hardware requirements. See System Requirements on page 13.

About this task

In console mode, the installation process is an interactive and text-based. You can install the product in the console mode only on the Ubuntu and RHEL operating systems.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the appropriate number during the installation of the product.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

If you do not want to configure the agent during the installation of the product, then you can leave the configuration fields blank. Later, when you decide to configure the agent, you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`/HCL/HCLoneTest/Majordomo.config`

Note: The **URL Alias** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Open the terminal and log in as a root user.
3. Run the following command to change the directory from root to the directory where you downloaded the installer zip file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/
4. Run the following command to extract the product installer zip file:

   ```
   unzip <product package name>
   ```

   For example, unzip HCL-OneTest-Performance-Agent-Windows-x64-v10.2.0.0.zip

   The command-line interface displays the list of preferred languages based on the display language that is set on your computer.

   Note: To identify the display language of your computer, the command-line interface displays an arrow mark (->) that precedes the name of the language.

5. Enter a number to select your preferred language, and then **Enter**.
   Result
   The installation instructions are displayed in your preferred language.
6. Run the following command to begin the installation process:

   ```
   ./<filename.bin> -i console
   ```

   For example, `./HCL-OneTest-Performance-Agent-Windows-x64-v10.2.0.0.bin -i console`
7. Review the license agreement and perform the following actions:
   - Read the license agreement by pressing any key along with the **Enter** key to continue reading the license agreement.
   - At any time, press `0 (zero)` along with the **Enter** key to go to the end of the license agreement.
   - After you read the license agreement, enter `Y` to accept the license agreement and continue with the installation.

Note: You can type `N` if you want to cancel the installation.

8. Provide the path where you want to install the product and perform the following steps:
    a. Press **Enter** to install the product in the default path: `/opt/HCL`.
    b. Enter the complete path of the directory and press **Enter** to install the product in the new directory path.

Note: You can type `Back` and press **Enter** to go to the previous step if you want to modify your inputs during the installation process.

9. **Optional:** Perform the following steps to configure the agent:

    a. Specify the values for the following parameters for Rational® Functional TesterHCL OneTest™ UI:

Field name	Description	Example
Host name	The hostname of Rational® Functional TesterHCL OneTest™ UI.	localhost
Port	The port number of Rational® Functional TesterHCL OneTest™ UI.	7080

    b. Specify the values for the following parameters for Rational® Test Automation ServerHCL OneTest™ Server:

Field name	Description	Example
Host name	The hostname of Rational® Test Automation ServerHCL OneTest™ Server. **Note:** ▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server must be resolvable through a Domain Name Server (DNS). ▪ An IP address cannot be the hostname of Rational® Test Automation ServerHCL OneTest™ Server.	localhost

Field name	Description	Example
	▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server through host files must not be specified. For example, `/etc/hosts` or `C:/Windows/system32/drives/etc/host`	
Port	The port number of Rational® Test Automation ServerHCL OneTest™ Server.	443
Token	An offline user token that is created from Rational® Test Automation ServerHCL OneTest™ Server.	eyJhbGciOiJIUzI1NiIsInR
URL Alias	The name of the **URL Alias** that you provided during the creation of the team space in Rational® Test Automation ServerHCL OneTest™ Server.	testteam

10. Review the pre-installation summary, and then press **Enter** to continue with the installation process.
11. Press **Enter** to exit the installer.

Result

Note: If the installation of the product is not successful, you can check the errors that occurred during the installation in the log file. The log file is available at the location: `/opt/HCL/Logs`.

Results

You have installed the agent on your computer.

Note: If the installation of the product is not successful, you can check the errors that occurred during the installation in the log file. The log file is available at the location: `/opt/HCL/Logs`.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Creating the properties file on Linux for Rational® Performance TesterHCL OneTest™ Performance Agent

To install the product in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 267 through 4 on page 267.
- Copying the contents of the sample properties file to your local properties file. See 5 on page 267.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as a root user.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/

4. Run the following command to generate the properties file:

   ```
   <installer_file.bin> -r <location of the installer.properties file>
   ```

   For example,

   For example, HCL-OneTest-Performance-Agent-Linux-x64-v10.1.0.0.bin -r /root/Downloads/installer.properties

5. Create your properties file by copying the content of the following sample properties file:

   ```
   # Replay feature output
   # ---------------------
   # This file was built by the Replay feature of Installer.
   # It contains variables that were set by Panels, Consoles or Custom Code.



   #Destination Folder
   #-------------------
   USER_INSTALL_DIR=/opt/HCL

   #HCL Load Generation Agent Configuration
   #---------------------------------------
   WORKBENCH_HOSTNAME=<host name of HCL OneTest UI>
   WORKBENCH_PORT=7080
   SERVER_HOSTNAME=<host name of server>
   SERVER_PORT=6969
   SERVER_TOKEN=<server token>
   SERVER_URL_ALIAS=<name of the URL alias for the team space>
   ```

   ```
   # Replay feature output
   # ---------------------
   # This file was built by the Replay feature of Installer.
   # It contains variables that were set by Panels, Consoles or Custom Code.
   ```

```
#Destination Folder
#------------------
USER_INSTALL_DIR=/opt/HCL

#HCL Load Generation Agent Configuration
#---------------------------------------
WORKBENCH_HOSTNAME=<host name of HCL OneTest Performance>
WORKBENCH_PORT=7080
SERVER_HOSTNAME=<host name of server>
SERVER_PORT=6969
SERVER_TOKEN=<server token>
SERVER_URL_ALIAS=<name of the URL alias for the team space>
```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing Agent in the silent mode on Linux

You can install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on different computers to apply load on the server that hosts the application under test or to run multiple Web UI tests. When you want to run the test scripts on any computer, you must install Rational® Performance TesterHCL OneTest™ Performance Agent.

Before you begin

You must have completed the following tasks:

- Uninstalled the previous version of the product. See Uninstalling Agent in the silent mode on Linux on page 290.
- Installed Konsole on your computer. The Konsole application is required to enable the shortcut icon of the product after installation. You can install Konsole by running the apt-get install konsole command in the terminal.

  Note: You need not install Konsole on the computer that is running RHEL 8 or later.

- Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.
- Created a properties file that contains the inputs for the installer during the installation process. See Creating the properties file on Linux for Rational Performance TesterHCL OneTest Performance Agent on page 266.
- Verified the software and hardware requirements. See System Requirements on page 13.

- Verified that the following libraries are available on your computer, if you want to install Rational® Functional TesterHCL OneTest™ UI:
  - libXp.so.6
  - libXm.so.4

Note: For more information on the library dependencies on Linux, refer to Library dependency on Linux.

About this task

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

If you decided to configure the agent, later, then you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`/HCL/HCLoneTest/Majordomo.config`

Note: The **SERVER_URL_ALIAS** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Open the terminal and log in as a root user.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <path of the downloaded folder>
   ```

   For example, cd Downloads/
4. Run the following command to extract the downloaded zip file:

   ```
   unzip <product package name>
   ```

   For example, unzip HCL-OneTest-Performance-Agent-Linux-x64-v10.1.0.0.zip
5. Run the following command to install the product:

   ```
   <installer_name> -i silent -l <language_code> -f <location_of_the_installer.properties_file_name>
   ```

   For example, you can run the following command to install Rational® Performance TesterHCL OneTest™ Performance Agent 10.2.0 in French:

   HCL-OneTest-Performance-Agent-Linux-x64-v10.2.0.0.bin -i silent -l fr -f /root/Downloads/installer.properties

The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-l** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` file to know more about the problem that occurred during the installation process. The error log is located at `/<installation directory>/Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `/<installation directory>/Logs`.

Results

You have installed the product on your computer.

Installation of the Agent on macOS

You can find information about installing Rational® Performance TesterHCL OneTest™ Performance Agent on macOS by using the stand-alone installer.

You can install the Agent by using any of the following methods:

- GUI mode
- Silent mode

Installing Agent in the GUI mode on macOS

You can install IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on different computers to apply load on the server that hosts the application under test or to run multiple Web UI tests. When you want to run the test scripts on mac machine, you must install Rational® Performance TesterHCL OneTest™ Performance Agent.

Before you begin

- You must have elevated privileges access.
- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling Agent in the GUI mode on macOS on page 292.
  - Verified the software and hardware requirements. See System Requirements on page 13.

About this task

All 32-bit test extensions such as SAP, Citrix, and Siebel are supported only for 32-bit Rational® Performance TesterHCL OneTest™ Performance Agent. The product installer does not support the following features:

Note: When you run a test or schedule by using Rational® Performance TesterHCL OneTest™ Performance Agent on macOS, by default, the majordomo.log file is stored in the $TMPDIR.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

If you do not want to configure the agent during the installation of the product, then you can leave the configuration fields blank. Later, when you decide to configure the agent, you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`/Application/HCL/HCLOnetest/Majordomo.config`

Note: The **URL Alias** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.
2. Extract the downloaded product installer file.
3. Double-click the installer to initiate the installation process.

   The GUI window is displayed.
4. Initiate the GUI installation from the command line by completing the following steps:

   a. Change the directory to where you have downloaded the installer by running the following command:

   ```
   cd <directory path>
   ```

   By default, the package is downloaded to `/Users/Applications/HCL`. For example, `cd HCL-OneTest-UI-Mac-v10.1.2.app/Contents/MacOS/cd HCL-OneTest-Performance-Mac-v10.1.2.app/Contents/MacOS/`

   b. Run the following command to initiate the product installation:

   ```
   ./<product name>
   ```

   For example, `./HCL-OneTest-UI-Mac-v10.1.2./HCL-OneTest-Performance-Mac-v10.1.2`

5. Select the language from the drop-down list to view the installation instructions and the product in your preferred language.

   The drop-down list displays the languages based on the display language that is set on your computer.

6. Read through the details on the **Introduction** window, and then click **Next**.
7. Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
8. Browse for the location or directory where you want to install the product, and then click **Next**.

   Note: By default, the product is installed at the following location: `/Applications/HCL`.

9. **Optional:** Perform the following steps to configure the agent:

   a. Specify the values for the following parameters for Rational® Functional TesterHCL OneTest™ UI:

Field name	Description	Example
Host name	The hostname of Rational® Functional TesterHCL OneTest™ UI.	localhost
Port	The port number of Rational® Functional TesterHCL OneTest™ UI.	7080

   b. Specify the values for the following parameters for Rational® Test Automation ServerHCL OneTest™ Server:

Field name	Description	Example
Host name	The hostname of Rational® Test Automation ServerHCL OneTest™ Server. **Note:** ▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server must be resolvable through a Domain Name Server (DNS). ▪ An IP address cannot be the hostname of Rational® Test Automation ServerHCL OneTest™ Server.	localhost

Field name	Description	Example
	▪ The hostname of Rational® Test Automation ServerHCL OneTest™ Server through host files must not be specified. For example, `/etc/hosts` or `C:/Windows/system32/drives/etc/host`	
Port	The port number of Rational® Test Automation ServerHCL OneTest™ Server.	443
Token	An offline user token that is created from Rational® Test Automation ServerHCL OneTest™ Server.	eyJhbGciOiJIUzI1NiIsInR
URL Alias	The name of the **URL Alias** that you provided during the creation of the team space in Rational® Test Automation ServerHCL OneTest™ Server.	testteam

10. Read through the installation details and, then click **Install**.
11. Click **Done** after the installation of the product is complete.

Results

You have installed the agent on your computer.

Creating the properties file on macOS for Rational® Performance TesterHCL OneTest™ Performance Agent

To install the product in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 274 through 4 on page 275 .
- Copying the contents of the sample properties file to your local properties file. See 5 on page 275.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by

using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Right-click the **Command Prompt** application and select **Run as Administrator**.
3. Run the following command to change the directory to a location where you downloaded the installer file:

```
cd <directory path>
```

   For example, cd C:\users\Downloads
4. Run the following command to generate the properties file:

```
<installer_file.exe> -r <location of the installer.properties file>
```

   For example, HCL-OneTest-Performance-Agent-AMC-x64-v10.1.0.0.exe -r C:\users\Downloads \installer.properties
5. Create the properties file by copying the content of the following sample properties file:

```
# Replay feature output
# ---------------------
# This file was built by the Replay feature of Installer.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Destination Folder
#------------------
USER_INSTALL_DIR=/Applications/HCL

#HCL Load Generation Agent Configuration
#---------------------------------------
WORKBENCH_HOSTNAME=<host name of HCL OneTest UI>
WORKBENCH_PORT=7080
SERVER_HOSTNAME=<host name of server>
SERVER_PORT=6969
SERVER_TOKEN=<server token>
SERVER_URL_ALIAS=<name of the URL alias for the team space>
```

```
# Replay feature output
# ---------------------
# This file was built by the Replay feature of Installer.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Destination Folder
#------------------
USER_INSTALL_DIR=/Applications/HCL

#HCL Load Generation Agent Configuration
#---------------------------------------
WORKBENCH_HOSTNAME=<host name of HCL OneTest Performance>
WORKBENCH_PORT=7080
SERVER_HOSTNAME=<host name of server>
SERVER_PORT=6969
```

```
SERVER_TOKEN=<server token>
SERVER_URL_ALIAS=<name of the URL alias for the team space>
```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing Agent in the silent mode on macOS

To get started with IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on macOS, in addition to the GUI mode, you can also use the silent installation method to install the product.

Before you begin

- You must have elevated privileges access.
- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling Agent in the GUI mode on macOS on page 292
  - Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.
  - Verified the software and hardware requirements. See System Requirements on page 13.
  - Created a properties file that contains the inputs for the installer during the installation process. See Creating the properties file on macOS for Rational Performance TesterHCL OneTest Performance Agent on page 274.

About this task

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

If you decided to configure the agent, later, then you can update the parameter values in the `Majordomo.config` file. The `Majordomo.config` file is at the following location:

`/Application/HCL/HCLOnetest/Majordomo.config`

Note: The **SERVER_URL_ALIAS** parameter name is shown as a **slug** in the `Majordomo.config` file.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

2. Open the terminal and log in as an administrator.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd /Users/username/Downloads

4. Extract the downloaded installer file.
5. Run the following command to install the product:

   ```
   <installer_name> -i silent -l <language_code> -f <location_of_the_installer.properties_file_name>
   ```

   For example, you can install Rational® Performance TesterHCL OneTest™ Performance Agent in French by running the following command:

   HCL-OneTest-Performance-Agent-MAC-x64-v10.2.0.0.app -i silent -l fr -f /Users/Applications/HCL/installer.properties

   The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-I** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` file to know more about the problem that occurred during the installation process. The error log is located at `/<installation directory>/Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `/<installation directory>/Logs`.

Results

You have installed the product on your computer.

Installation of the product software on macOS

You can find information about installing the product software on macOS by using the stand-alone installer.

You can install the product software by using any of the following methods:

- GUI mode
- Silent mode

Installing the product in the GUI mode on macOS

To get started with the desktop client on macOS, you can install the product in the GUI mode by using the stand-alone installer.

Before you begin

- You must have elevated privileges access.
- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling the product in the GUI mode on macOS on page 291.
  - Verified the software and hardware requirements. See System Requirements on page 13.

About this task

You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the preferred language of the stand-alone installer and the product by selecting the language from the drop-down list during the installation of the product. The drop-down list displays the languages based on the default language that is set on your computer.

The following languages are displayed in the drop-down list based on the display language of your computer:

List of preferred languages that are displayed in the drop-down list based on the display language of your computer							
• English • French • German • Italian • Spanish • Portuguese Brazil	• Simplified Chinese • English	• Traditional Chinese • English	• Japanese • English	• Czech • English • Hungarian • Polish	• Korean • English	• Turkish • English	• Russian • English

For example, if the display language of your computer is *Japanese*, then the drop-down list shows *Japanese* and *English* as the preferred languages.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Extract the downloaded product installer file.
3. Double-click the installer to initiate the installation process.

   The GUI window is displayed.

4. Select the language from the drop-down list to view the installation instructions and the product in your preferred language.

The drop-down list displays the languages based on the display language that is set on your computer.

5. Read through the details on the **Introduction** window, and then click **Next**.
6. Read the license agreement carefully, select the **I accept** checkbox, and then click **Next**.
7. Browse for the location or directory where you want to install the product, and then click **Next**.



Note: By default, the product is installed at the following location: `/Applications/HCL`.

8. Read through the installation details and, then click **Install**.
9. Click **Done** after the installation of the product is complete.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Creating a properties file on macOS

To install the product in the silent mode, you must first create a properties file that contains the data required to install the product. Alternatively, you can generate the properties file by using the stand-alone installer.

About this task

You can create the properties file by using one of the following methods:

- Generating the properties file by using the stand-alone installer. See steps 1 on page 280 through 4 on page 280.
- Copying the contents of the sample properties file to your local properties file. See 5 on page 281.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as an administrator.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd /Users/username/Downloads
4. Run the following command to generate the properties file:

```
<installer_file.app> -r <location of the installer.properties file>
```

For example, HCL-OneTest-Performance-MAC-x64-v10.1.0.0.app -r /Users/Applications/HCL/installer.propertiesHCL-OneTest-UI-MAC-x64-v10.1.0.0.app -r /Users/Applications/HCL/installer.properties

5. Create the properties file by copying the content of the following sample properties file:

```
#Host Name
# Replay feature output
# ---------------------
# This file was built by the Replay feature of Installer.
# It contains variables that were set by Panels, Consoles or Custom Code.



#Destination Folder
#------------------
USER_INSTALL_DIR=/Applications/HCL
```

Results

You have generated or created the properties file.

What to do next

You can install the product by using the silent mode method.

Installing the product in the silent mode on macOS

To get started with the product, you can install the product on macOS by using the silent mode installation method.

Before you begin

- You must have elevated privileges access.
- You must have completed the following tasks:
  - Uninstalled the previous version of the product. See Uninstalling the product in the silent mode on macOS on page 291.
  - Deleted the `installer.properties` file generated from the previous installation if you are installing a newer version of the product.
  - Created a properties file that contains the inputs for the installer during the installation process. See Creating a properties file on macOS on page 280.
  - Verified the software and hardware requirements. See System Requirements on page 13.

About this task

You cannot update, modify, or roll back the product by using the stand-alone installer.

You can use the stand-alone installer and the product in your preferred language. The display language of your computer is selected as the preferred language of the stand-alone installer. For example, if the display language of your computer is set as French, then the preferred language of the stand-alone installer is also selected as French.

You can change the language of the stand-alone installer and the product by providing the **-l** *language_code* parameter during the installation of the product.

1. Download the product installer from the HCL® License & Delivery portal.

   You must download the appropriate product variant, version, and architecture based on your requirements.

   Note: When you want to install both Rational® Functional TesterHCL OneTest™ UI and Rational® Performance TesterHCL OneTest™ Performance on your computer, you can install the products by using a single shell-shared `HCL OneTest UI and Performance` installer that is available in the HCL® License & Delivery portal.

2. Open the terminal and log in as an administrator.
3. Run the following command to change the directory to a location where you downloaded the installer file:

   ```
   cd <directory path>
   ```

   For example, cd /Users/username/Downloads

4. Extract the downloaded the product installer file.
5. Run the following command to start the installation process:

   ```
   <installer_file.app> -i silent -l <language_code> -f <location of the properties file>
   ```

   For example, you can install the product in French by running the following command:

   HCL-OneTest-Performance-MAC-x64-v10.2.0.0.app -i silent -l fr -f /Users/Applications/HCL/installer.propertiesHCL-OneTest-UI-MAC-x64-v10.2.0.0.app -i silent -l fr -f /Users/Applications/HCL/installer.properties

   The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

Notes:

- The **-l** parameter is optional. If you do not want to use this parameter, then the following information are displayed in a language based on the display language that is set on your computer:
  - The installation instructions
  - The language of the product
- If the installation is terminated due to any reason, you must check the `Installerror.txt` to know more about the problem that occurred during the installation process. The error log is located at `/<installation directory>/Logs`.

Result

You can verify the status of the installation of the product by checking the log file at `\<installation directory>\Logs`.

Results

You have installed the product on your computer.

What to do next

You must enter the license key when you open the product. See License management on page 296.

Changing the language of the product on macOS

When you want to use the product in a different language other than the display language of your computer, then you can change the language of the product by using the command-line interface.

About this task

Important: After you changed the language and closed the application for any reason, the product displays in the language that is set in your computer if you re-open the product.

For example, consider the display language of your computer is set as French and you change the language of the product to Japanese by using the command-line interface. When you close the product and open it again, the product displays in the French language.

1. Open the terminal and log in as an administrator.
2. Run the following command to change the directory to a location where you installed the product:

   ```
   cd <directory path>
   ```

   For example, cd /Users/Applications/HCL/HCL-OneTest
3. Run the following command to change the language of the product:

```
./eclipse -nl <language_code>
```

Where *<language_code>* is the code of your preferred language.

The following table lists the language codes that you might want to choose during the installation:

Languages	Chinese (Simplified)	Chinese (Traditional)	Czech	English	French	German	Hungarian	Italian	Japanese	Korean	Polish	Portuguese (Brazil)	Russian	Spanish	Turkish
Language codes	zh_CN	zh_TW	cz	en	fr	de	hu	it	ja	ko	pl	pt_BR	ru	es	tr

For example, ./eclipse -nl ja

Result

The product opens in the Japanese language.

Results

You have changed the language of the product until you restart the product.

Uninstallation of the product by using the stand-alone installer

When you no longer require the product, you can use the stand-alone installer to uninstall the products that you have installed.

Uninstalling the product on Windows systems

You can find information about uninstalling the product on Windows system by using either the stand-alone installer.

You can uninstall the product by using any of the following methods:

- GUI mode
- Silent mode

Uninstalling the product in the GUI mode on Windows systems

When you want to remove the product on a Windows system computer, use the uninstall option. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. On the **Start** menu, click **Add or remove programs** option.
2. On the **Apps & Features** page, click the product software that you want to remove.
3. Click **Uninstall**.

   A dialog appears with the warning message.
4. In the dialog, click **Uninstall** and follow the on-screen instructions.

   The product is removed from your system.

Results

You have uninstalled the product from your computer.

Uninstalling the product in the silent mode on Windows systems

When you no longer require a version of the product, you can uninstall the product automatically without manual intervention by using the command prompt option.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Right-click the **Command Prompt** application and select **Run as Administrator**.
2. Change to the directory where the product is installed by running the following command:

   ```
   cd <directory path>
   ```

   For example, cd C:\Program Files\HCL\Uninstall
3. Start the uninstallation process by running the following command:

   ```
   Uninstall-HCL-OneTest-UI.exeHCL-OneTest-Performance.exe -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Uninstalling Agent in the GUI mode on Windows systems

When you want to remove IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on a Windows system computer, use the uninstall option. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. On the **Start** menu, click **Add or remove programs** option.
2. On the **Apps & Features** page, click the product software that you want to remove.
3. Click **Uninstall**.

   A dialog appears with the warning message.
4. In the dialog, click **Uninstall** and follow the on-screen instructions.

   The product is removed from your system.

Results

You have uninstalled the agent from your computer.

Uninstalling Agent in the silent mode on Windows systems

When you no longer require IBM® Rational® Performance TesterHCL OneTest™ Performance Agent, you can uninstall it automatically without manual intervention by using the command prompt option. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Right-click the **Command Prompt** application and select **Run as Administrator**.
2. Change to the directory where the product is installed by running the following command:

   ```
   cd <directory path>
   ```

   For example, cd C:\Program Files\HCL\Uninstall
3. Start the uninstallation process by running the following command:

   ```
   Uninstall-HCL-OneTest-Performance-Agent.exe -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Uninstalling the product on Linux

You can find information about uninstalling the product on Linux by using the stand-alone installer.

You can uninstall the product by using any of the following methods:

- GUI mode
- Console mode
- Silent mode

Uninstalling the product in the GUI mode on Linux

When you want to remove the product on Linux machine, use the uninstall option. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as root user.
2. Click **Applications > Programming > Uninstall**.
3. On the **Uninstall** screen, click **Uninstall** and follow the on-screen instructions.
4. Uninstall the product by running the following commands in the terminal, if you are a non-root user:

```
<installation directory>/Uninstall
sudo ./Uninstall-HCL-OneTest-UI
```

Results

You have uninstalled the product from your computer.

Uninstalling the product in the console mode on Linux

When you no longer want to use the product installed on Ubuntu or Red Hat Enterprise Linux (RHEL) operating system, you can uninstall the product in the console mode.

Before you begin

- You must be a root user.
- You must close browsers and other applications that are enabled by the product before you uninstall the product.

1. Open a terminal and change the current directory to the installation directory by running the following command:

   ```
   cd <directory path>/<installation folder>/Uninstall
   ```

   For example, `cd /opt/HCL/OneTest-UI/Uninstallcd /opt/HCL/OneTest-Performance/Uninstall`

2. Run the following command to uninstall the product:

   ```
   ./Uninstall-HCL-OneTest-UIPerformance -uninstall -i console
   ```

3. Read the details about the uninstallation process, and then press **Enter**.
4. Exit the console mode by pressing **Enter** after the product is uninstalled.

Results

You have uninstalled the product from your computer.

What to do next

You can reinstall the product when you want to upgrade the product to the latest version. You can also reinstall the product if the previous installation is unsuccessful. See Installing the product in the console mode on page 242.

Uninstalling the product in the silent mode on Linux

When you no longer require a version of the product, you can uninstall the product automatically without manual intervention by using the command prompt option.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as a root user.
2. Change the directory to where you have installed the product by entering the following command:

   ```
   cd <directory path>
   ```

   For example, cd /opt/HCL/Uninstall-HCL-OneTest-PerformaceHCL-OneTest-UI

3. Initiate the uninstallation process by entering the following command:

   ```
   ./Uninstall-HCL-OneTest-PerformanceHCL-OneTest-UI -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Uninstalling Agent in the GUI mode on Linux

When you want to remove IBM® Rational® Performance TesterHCL OneTest™ Performance Agent on Linux machine, use the uninstall option.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as root user.
2. Click **Applications > Programming > Uninstall**.
3. On the **Uninstall** screen, click **Uninstall** and follow the on-screen instructions.
4. Uninstall the product by running the following commands in the terminal, if you are a non-root user:

```
<installation directory>/Uninstall
sudo ./Uninstall-HCL-OneTest-UI
```

Results

You have uninstalled the agent from your computer.

Uninstalling Agent in the console mode on Linux

When you no longer want to use IBM® Rational® Performance TesterHCL OneTest™ Performance Agent installed on Ubuntu or Red Hat Enterprise Linux (RHEL) operating system, you can uninstall IBM® Rational® Performance TesterHCL OneTest™ Performance Agent in the console mode.

Before you begin

- You must be a root user.
- You must close browsers and other applications that are enabled by the product before you uninstall the product.

1. Open a terminal and change the current directory to the installation directory by running the following command:

   ```
   cd <directory path>/<installation folder>/Uninstall
   ```

   For example, `cd /opt/HCL/OneTest-Performance-Agent/Uninstall`
2. Run the following command to uninstall the product:

   ```
   ./Uninstall-HCL-OneTest-Performance-Agent -uninstall -i console
   ```
3. Read the details about the uninstallation process, and then press **Enter**.
4. Exit the console mode by pressing **Enter** after the product is uninstalled.

Results

You have uninstalled the agent from your computer.

What to do next

You can reinstall the product when you want to upgrade the product to the latest version. You can also reinstall the product if the previous installation is unsuccessful. See Installing Agent in the console mode on Linux on page 262.

Uninstalling Agent in the silent mode on Linux

When you no longer require Rational® Performance TesterHCL OneTest™ Performance Agent, you can uninstall the product automatically without manual intervention using the command prompt option. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as a root user.
2. Change the directory to where you have installed the product by entering the following command:

   ```
   cd <directory path>
   ```

   For example, cd /opt/HCL/Uninstall-HCL-OneTest-Performace-Agent
3. Initiate the uninstallation process by entering the following command:

   ```
   ./Uninstall-HCL-OneTest-Performance-Agent -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Uninstalling the product on macOS

You can find information about uninstalling the product on macOS by using either the stand-alone installer.

You can uninstall the product by using any of the following methods:

- GUI mode
- Silent mode

Uninstalling the product in the GUI mode on macOS

When you no longer require the product, you can uninstall the product in the GUI mode by using the stand-alone installer. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Go to the directory where the product is installed.
2. Double-click the `Uninstall-HCL-OneTest-UI` file and follow the on screen instructions.

   The product is uninstalled from your computer.

Results

You have uninstalled the product from your computer.

Uninstalling the product in the silent mode on macOS

When you no longer require a version of the product, you can uninstall it by using the silent mode method. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as an administrator.
2. Change to the directory where the product is installed by entering the following command:

   ```
   cd <directory path>
   ```

   For example, cd Applications/HCL/Uninstall-HCL-HCL-OneTest-PerformanceHCL-OneTest-UI/Contents/MacOS
3. Start the uninstallation process by running the following command:

   ```
   /Uninstall-HCL-OneTest-PerformanceHCL-OneTest-UI -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Uninstalling Agent in the GUI mode on macOS

When you no longer require IBM® Rational® Performance TesterHCL OneTest™ Performance, you can uninstall the product in the GUI mode by using the stand-alone installer.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Go to the directory where the product is installed.
2. Double-click the `Uninstall-HCL-OneTest-UI` file and follow the on screen instructions.

   The product is uninstalled from your computer.

Results

You have uninstalled the agent from your computer.

Uninstalling Agent in the silent mode on macOS

When you no longer require IBM® Rational® Performance TesterHCL OneTest™ Performance Agent, you can uninstall it in the silent mode. You can choose to uninstall the product by using any one of the uninstallation methods regardless of the mode of installation.

Before you begin

You must have completed the following tasks:

- Closed any open windows of the product.
- Closed any open web browsers.
- Closed all the other applications that are enabled by the product.

1. Log in as an administrator.
2. Change to the directory to where the product is installed by entering the following command:

   ```
   cd <directory path>
   ```

   For example, cd Applications/HCL/Uninstall-HCL-HCL-OneTest-Performance-Agent/Contents/MacOS
3. Start the uninstallation process by running the following command:

   ```
   /Uninstall-HCL-OneTest-Performance-Agent -uninstall -i silent
   ```

Results

You have uninstalled the product from your computer.

Upgrading and migrating

When you want to use the enhanced functionality of Rational® Functional TesterHCL OneTest™ UI, you must upgrade to the latest version of the product software.

Remember:

- You can upgrade to Rational® Functional TesterHCL OneTest™ UI 10.5.3 from the previous versions, 10.5.2, 10.5.1, and 10.5.0. See Updating Rational Functional TesterHCL OneTest UI on page 293.
- If you installed Rational® Functional TesterHCL OneTest™ UI that is earlier than 10.5.0, then you cannot upgrade the product to the latest version. You must uninstall the existing version of the product before you install the latest version of the product.
- After you install Rational® Functional TesterHCL OneTest™ UI 10.5.3, at any point in time if you want to use the previous version of the product, you cannot roll back to the previous version. If you want to use the previous version of the product, you must uninstall the existing version, and then install the required version of the product.

Updating Rational® Functional TesterHCL OneTest™ UI

For some releases, you can install updates for packages that were installed with Installation Manager. Package updates provide fixes and updates to installed features and might also include new features that you can install using the Modify Packages wizard.

Before you begin

By default, Internet access is required unless your repository preferences points to your local update site.

Each installed package has the location embedded for its default IBM® update repository. For Installation Manager to search the IBM® update repository locations for the installed packages, the preference **Search service repositories during installation and updates** on the Repositories preference page must be selected. This preference is selected by default.

See the Installation Manager help for more information.

Note:

- Close all programs that were installed using Installation Manager.
- Close Eclipse, as well as any open web browsers, and all other applications that are enabled by Rational® Functional TesterHCL OneTest™ UI.
- Close the Eclipse and Visual Studio IDEs, as well as any open web browsers, and all other applications that are enabled by Rational® Functional TesterHCL OneTest™ UI.

About this task

You can update your software in two ways:

- Online mode: This method requires an internet connection. Installation Manager connects to the IBM update repositories that are preconfigured when the product is installed, and downloads and installs the update package.
- Offline mode: While connected to the internet, download the package from the IBM update repository and extract the files to a temporary location. Then in offline mode, run Installation Manager and update the installation.

1. From the Start page of the Installation Manager, click **Update**.
2. If IBM® Installation Manager is not detected on your system or if an older version is already installed, then you must continue with the installation of the latest release. Follow the instructions in the wizard to complete the installation of IBM® Installation Manager
3. In the Update Packages wizard, select the location of the package group where the Rational® Functional TesterHCL OneTest™ UI product package you want to update is installed or select the **Update All** check box, and then click **Next**.
   Installation Manager searches for updates in its repositories and the predefined update sites for Rational® Functional TesterHCL OneTest™ UI. A progress indicator shows the search is taking place.
4. If updates for a package are found, then they are displayed in the **Updates** list on the Update Packages page under the selected package. Only recommended updates are displayed by default. Click **Show all** to display all updates found for the available packages.
   a. To learn more about an update, click the update and review its description under **Details**.
   b. If additional information about the update is available, a **More info** link will be included at the end of the description text. Click the link to display the information in a browser. Review this information before installing the update.
5. Select the updates that you want to install or click **Select Recommended** to restore the default selections. Updates that have a dependency relationship are automatically selected and cleared together.
6. Click **Next** to continue.
7. On the Licenses page, read the license agreements for the selected updates. On the left side of the **License** page, the list of licenses for the updates you selected is displayed; click each item to display the license agreement text.

a. If you agree to the terms of all the license agreements, click **I accept the terms of the license agreements**.
b. Click **Next** to continue.

8. On the Features page, select the package features that you want to install or remove.
   a. To learn more about a feature, click the feature and review the brief description under **Details**.
   b. If you want to see the dependency relationships between features, select **Show Dependencies**. When you click a feature, any features that depend on it and any features that are its dependents are shown in the Dependencies window. As you select or exclude features in the packages, Installation Manager will automatically enforce any dependencies with other features and display updated download size and disk space requirements for the installation.
9. On the Summary page, review your choices before installing the updates.
   a. If you want to change the choices you made on previous pages, click **Back**, and make your changes.
   b. When you are satisfied, click **Update** to download and install the updates. A progress indicator shows the percentage of the installation completed.



Note: During the update process, Installation Manager might prompt you for the location of the repository for the base version of the package. If you installed the product from CDs or other media, they must be available when you use the update feature.

10. **Optional:** When the update process completes, a message that confirms the success of the process is displayed near the top of the page. Click **View log file** to open the log file for the current session in a new window. You must close the Installation Log window to continue.
11. Click **Finish** to close the wizard.
12. **Optional:** Only the features that you already have installed are updated using the **Update** wizard. If the update contains new features that you would like to install, run the **Modify** wizard and select the new features to install from the feature selection panel.

Migrating test assets from earlier versions of Functional Tester

Follow these instructions to learn about migrating test assets from earlier versions of Functional Tester.

Enabling

Before you upgrade Functional Tester, close the Eclipse and Visual Studio IDEs, as well as any open web browsers, and all other applications that are enabled by Functional Tester.

As part of the upgrade, the first time you run the Enabler you will get a message informing you that Java™ environments that were previously enabled will be automatically disabled. You must then enable Java environments that you want to use for testing. Web browsers that were previously enabled will remain that way. Click **Configure > Enable Environments for Testing** on the product menu to run the Enabler. For more information on using the Enabler, see Enabling Java™ Environments on page 778. You must disable the next generation plug-in for existing JREs associated with browsers before upgrading.

Migrating test assets from earlier versions of Rational® Functional TesterHCL OneTest™ UI

All test assets from earlier versions of Rational® Functional TesterHCL OneTest™ UI, including projects, scripts, object maps, and verification points, work with the current version of the product. However, scripts that you record with the current version of the product will not play back on earlier versions. When you play back a script that was recorded with a previous version, a warning message is displayed in the log file. To view the log file without the warning message, you must disconnect the project, and connect again.

To connect to the project:

1. Right-click the functional test project, and click **Disconnect Project**. The project is removed from the **Functional Test Projects** view.
2. Click **File > Connect to a Functional Test Project**.
3. Click **Browse** to browse to the project location path. The project name is displayed in the **Project name** field when you browse to the project location path.
4. Click **Finish**. A dialog box prompting you to upgrade the project from the older version and connect again is displayed.
5. Click **Yes** to confirm. Click **Cancel** if you want to disconnect from the project.

License management

Licensing for IBM® Rational® Functional TesterHCL OneTest™ UI is administered through HCL Software License & Download portal. This portal is a FlexNet-based web application that helps to manage software entitlements and licenses.

When a software order is placed and acknowledged, a software entitlement is created. You must then follow the instructions in the Software Order Acknowledgment document that you receive to activate your entitlement, create devices, and download the software from the portal.

The license portal provides both software distribution and management of your software entitlements that are purchased from HCL Software. The portal provides control and flexibility on how to consume your licenses. An organization identifies one of its resources as a License Manager (also called Tech or Portal Admin) who is familiar with the language of licenses.

For more information about the HCL Software License & Download portal, you can refer to the following knowledge articles:

- What is the HCL Software License & Download portal (FlexNet portal)?
- How to find HCL Product Releases in HCL Software License & Download portal
- Managing Users on the HCL Software License & Download portal

If you do not have access to the internet, you can install and configure a Local License Server (LLS).See Configuring licenses by using a Local License Server on page 299.

License description

You require floating seat licenses to use IBM® Rational® Functional TesterHCL OneTest™ UI. If you want to run AFT Suites or parallel execution of tests on more than 5 channels, you require execution licenses.

When you place an order for a software application, the HCL License Fulfilment team acknowledges the order and creates a software entitlement for you. You can then create devices and map the software entitlement with the devices through the HCL® License & Delivery portal. Every device is associated with a server ID. This server ID is applied to the product. The team can create multiple software entitlements based on your requirements.

Note: If you use the Local License Server, you must also map the software entitlements with the Local License Server.

You can use any of the following licenses according to your requirements:

- **Floating seat license to use IBM® Rational® Functional TesterHCL OneTest™ UI**

  To use IBM® Rational® Functional TesterHCL OneTest™ UI, you require a floating seat license or you can also use a IBM® Rational® Test WorkbenchHCL OneTest™ Studio floating license. A floating seat license is in the checked-out state when you use IBM® Rational® Functional TesterHCL OneTest™ UI, and the license is in the checked-in state in the HCL® License & Delivery portal when the license is not used for 15 minutes. Multiple users can use IBM® Rational® Functional TesterHCL OneTest™ UI by using floating seat licenses. However, the total number of concurrent users cannot exceed the number of floating licenses that you purchase.

- **Floating license to run AFT Suites/tests**

  To run AFT Suites or parallel execution of tests on more than 5 channels, you require consumption-based licenses, execution floating licenses, or both.

  When you use consumption-based licenses, an appropriate number of licenses are in the checked-out state when you start the test run and these licenses are permanently consumed after use. The number of licenses that are checked out depend on the configuration of the AFT Suites/tests.

  The execution floating licenses are different from the floating seat licenses, but both the licenses work in a similar manner. A number of floating licenses are in the checked-out state when you start the test run, and the license is in the checked-in state in the HCL® License & Delivery portal within 15 minutes of the completion of test run. You can reuse execution floating licenses for an infinite number of times until the license expiration date.

License configuration

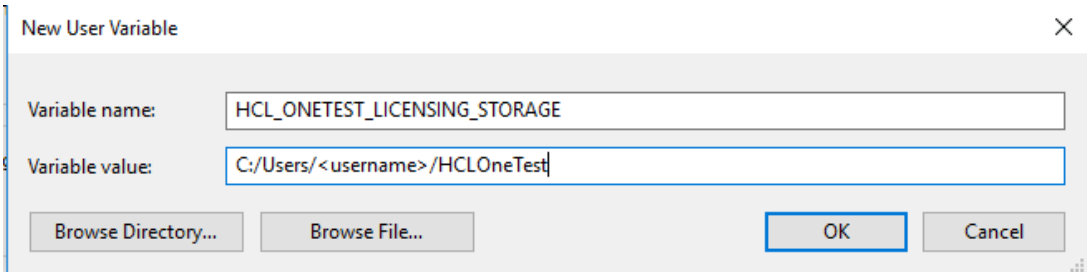
You must first apply a license to start using the product.

When you start the product for the first time, a licensing dialog box is displayed. Specify the server ID that was provided to you or copy the ID from the HCL® License & Delivery portal. When you submit the server ID, the product connects with the HCL® License & Delivery portal to verify it and if there is a license available, it is checked out so

that you can use the product. If the license is not available, a message is displayed about it. In most cases, you must not change the server URL in the licensing dialog.

Notes:

- If the license is not used for 15 minutes, the license is returned to the server for others to consume it. If the product looses connectivity to the HCL® License & Delivery portal, you can use the product for two hours only if the server ID that you entered last time is correct.
- After the product installation, when you apply the license to the product, the license information is cached in a system directory. If there are any permission issues accessing the directory or the directory is deleted, there can be licensing error when starting the product. The workaround is to create an environment variable *HCL_ONETEST_LICENSING_STORAGE* and specify a directory path. The licensing information is now stored in the new directory. For example,



You can also apply the license after you open the product by using any of the following methods based on your requirement:

- Configuring licenses by using a cloud-based License Server on page 299
- Configuring licenses by using a Local License Server on page 299
- Configuring licenses by using a Proxy Server on page 301

To apply the license without opening the product, add the licensing parameters as environment variables:

Variables	Values
HCL_ONETEST_LICENSING_URL	https://hclsoftware.compliance.flexnetoperations.com
HCL_ONETEST_LICENSING_ID	Enter your server ID. The ID is a 12-character alphanumeric identifier.

When running the tests from command line, you do not have to specify any licensing argument. However, when running a schedule for IBM® Rational® Performance TesterHCL OneTest™ Performance or accelerated/distributed tests for IBM® Rational® Functional TesterHCL OneTest™ UI in an uninterrupted mode, you must specify `vmargs`

`-Dhptcostconfirm` argument in the command. The uninterrupted mode refers to any of the non-GUI based test execution scenarios such as the following use cases:

- Running a test from the command line
- Running a test with Jenkins
- Running a test with Ant
- Running a test with UrbanCode™ Deploy
- Running a test with IBM® Engineering Test Management

Configuring licenses by using a cloud-based License Server

You can apply license by using a cloud-based License Server to use Rational® Functional TesterHCL OneTest™ UI.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click **Windows > Preferences > Test > HCL Licensing**.
3. Select the **Server Type** as **Cloud**.
4. Enter the URL of the License Server in the **Server URL** field.
5. Enter the ID of the License Server in the **Server ID** field.
6. Click **Test Connection** to verify the connection to the License Server.
7. Click **Apply** to configure the license.

Results

You have configured the license to use Rational® Functional TesterHCL OneTest™ UI. The **HCL Licensing** dialog box displays the following information:

- The number of Virtual Users (VUs)
- The number of VU Execution hours
- The number of floating licenses

What to do next

You can work with Rational® Functional TesterHCL OneTest™ UI.

Configuring licenses by using a Local License Server

You might not be able to connect your computer to the internet due to certain restrictions. In such cases, you can configure a Local License Server (LLS) behind a firewall to use Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Installed and started the LLS. For more information, refer to the HCL Software License & Download portal documentation.
- Imported a self-signed certificate into OneTest Java key store if you installed the LLS by using the self-signed certificate and used an HTTPS protocol to check out licenses from the LLS.

  Note: You can run the following commands to import the certificate into OneTest Java key store:

  ```
  cd <install_dir>\HCL\HCLOneTest\jdk\bin
  keytool -printcert -sslserver <LLS_host>:<LLS_port> -rfc | keytool -import -noprompt
   -alias hclfnls -keystore "<install_dir>\HCL\HCLOneTest\jdk\jre\lib\security\cacerts"
   -storepass changeit
  ```

  You must replace `<LLS_host>` and `<LLS_port>` with the hostname and port number of the LLS.

  Important: If you used an HTTP protocol to check out licenses from the LLS, you do not require to import the self-signed certificate.

- Mapped entitlements of software with the LLS to serve your requests. For more information, refer to the HCL Software License & Download portal documentation.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click **Windows > Preferences > Test > HCL Licensing**.
3. Select the **Server Type** as **Local**.
4. Replace the URL of the cloud-based License Server with the URL of your LLS in the **Server URL** field.

   Remember: You must enter the URL of the LLS in the following format:

   ```
   http://myserver:portNumber/request
   ```

5. Click **Test Connection** to verify the connection to the LLS.
6. Click **Apply** to configure the license.

Results

You have configured the license to use Rational® Functional TesterHCL OneTest™ UI. The **HCL Licensing** dialog box displays the following information:

- The number of Virtual Users (VUs)
- The number of VU Execution hours
- The number of floating licenses

What to do next

You can work with Rational® Functional TesterHCL OneTest™ UI.

Configuring licenses by using a Proxy Server

You can connect directly to a cloud-based license server when you start Rational® Functional TesterHCL OneTest™ UI for the first time. As an alternative, you can configure Rational® Functional TesterHCL OneTest™ UI to use a Proxy Server.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click **Windows > Preferences > Test > HCL Licensing**.
3. Select the **Server Type** as **Cloud** or **Local**.
4. Select the **Use Proxy Server** checkbox and specify the **Host** and **Port** of the Proxy Server.
5. **Optional:** Select the **Provide Credentials** checkbox and specify the **User Name** and **Password** if login credentials are required to access the Proxy Server.
6. Click **Test Connection** to verify the connection to the Proxy Server.
7. Click **Apply** to configure the license.

Results

You have configured the license to use Rational® Functional TesterHCL OneTest™ UI. The **HCL Licensing** dialog box displays the following information:

- The number of Virtual Users (VUs)
- The number of VU Execution hours
- The number of floating licenses

What to do next

You can work with Rational® Functional TesterHCL OneTest™ UI.

Collecting usage metrics data

To provide an insight into the usage patterns of the product itself and not necessarily the application that it is testing, the usage metrics are collected. The product can collect the metrics about the usage of the tool including and not limited to the number of tests executed, the number of actions performed against an application under test, the number of successful versus failed actions, verification points, and so on.

Before you begin

- Ensure that IBM® Rational® Test Control PanelHCL® Quality Server is installed. See the Rational® Test Control PanelHCL® Quality Server documentation for the installation instructions.
- Ensure that the license key is applied. See License configuration on page 297.

About this task

When you collect usage metrics, the data is collected for internal use to provide you better services. For example, in a Web UI test, all user actions such as click, enter text, and so on are counted and collected.For example, in an HTTP

test, the number of HTTP Pages and Requests are counted and logged. No information about the application under test is collected.

1. Click **Window > Preferences > Test > Rational Test Control Panel > HCL Quality Server**.
2. Specify the IP address or host name of the computer where Rational® Test Control PanelHCL® Quality Server server is installed.

   Note: If the IP address or host name is not specified, depending on the license key setting, a warning message might be logged every time you run a test, the test execution itself might be blocked, or the tests can be executed without any warning messages.

3. Click **Test Connection** to check whether the connection is established. Click **OK** if the connection is established.

License management

Licensing for your installed IBM® software and customized packages is administered through the Manage Licenses wizard in the IBM® Installation Manager. The Manage Licenses wizard displays license information for each installed package.

Using the Manage Licenses wizard, you can apply either a license to a product or upgrade trial versions of an offering to a licensed version.

For more information about managing licenses for your Rational® product, see these resources:

- Rational licensing support
- IBM Common Licensing documentation
- Configuring the License Key Server

License descriptions

You can understand the types of licenses available for the product and choose the correct license to run tests.

IBM® Rational® Functional TesterHCL OneTest™ UI requires two types of licenses: product activation and runtime licenses. You need product activation licenses such as activation kits, or floating/token licenses to run the product. Runtime licenses are unique to Rational® Functional TesterHCL OneTest™ UI and they consist of virtual tester pack floating license keys.

As a purchaser of an IBM®HCL® Rational® software product, you can choose from the following type of product licenses:

- Activation kits on page 303
- Floating licenses on page 303

- Token licenses on page 303
- Runtime licenses – Virtual tester pack floating license keys on page 304

You manage product activation by using Installation Manager. You manage runtime licenses by using IBM® Rational® License Key Administrator.

The best choice of licenses for your organization depends upon how many people use the product, how often they require access, and how you prefer to purchase software.

Activation kits

Activation kits are product and version specific (they are not host-specific). You simply need to obtain and install an "activation kit" for your product. The activation kits for the most recent product releases can be downloaded in Passport Advantage Online from the software download pages of the company account that has the licenses in active maintenance. For more information, refer to Importing a product activation kit.

Floating licenses

An IBM®HCL® Rational® Floating license is a license for a single software product that can be shared among multiple team members; however, the total number of concurrent users cannot exceed the number of floating licenses you purchase. For example, if you purchase one floating license for a Rational® software product, any user in your organization can use the product at any given time. Another person who wants to access the product must wait until the current user logs off.

To use floating licenses, you must obtain floating license keys and install them on a Rational® License Server. The server responds to user requests for access to the license keys; the server grants access to the number of concurrent users that equals the number of licenses the organization purchased. For more information about the Rational® License Server, see Managing licenses with Rational® Common Licensing.

Floating license enforcement provides these benefits:

- License compliance enforcement throughout the organization
- Fewer license purchases
- License keys served for IBM®HCL® Rational® Team Unifying and IBM® Rational® Software Delivery Platform desktop products from the same license server
- Some versions of Rational® products require an upgraded version of the Rational® License Server. See this support article for license upgrade information: http://www.ibm.com/support/docview.wss?uid=swg21250404

Token licenses

The token-based license model means that you can buy a certain number of token licenses. If you use a Rational® tool that checks out a feature that is token-based, the feature line in the license file specifies the number of tokens that are checked out. Token-based licenses can only be used with floating licenses. For more details about token licensing, contact your local IBM®HCL® marketing representative.

Runtime licenses – Virtual tester pack floating license keys

In addition to product activation, you need runtime licensing when you run AFT suites or parallel execution of tests on more than 5 channels. Runtime licensing consists of the virtual tester pack floating license keys. You can check out these keys from the same Rational® License Server that you use for floating license product activation.

Runtime licenses enablement for testing

You can check and enable the correct runtime license to run AFT suites or parallel execution of tests on more than 5 channels.

To check out virtual tester pack floating license keys in Windows™, you must use the IBM® Rational® License Key Administrator program to point to a Rational® License Server. Use the version of IBM® Rational® License Key Administrator that was installed with Rational® Functional TesterHCL OneTest™ UI.

For information about IBM® Rational® License Key Administrator v 9.0, see Configuring a client by using License Key Administrator (LKAD).

There is no Linux™ version of the IBM® Rational® License Key Administrator. To check out virtual tester pack floating license keys in Linux™, you must create and edit the `.flexlmrc` file. Create the file `.flexlmrc` in the home directory of a user who runs parallel execution with more than five channels, or uses the floating license product activation. Edit the `.flexlmrc` file, adding a line with the computer name or the IP address of the Rational® License Server. For example, add the line `RATIONAL_LICENSE_FILE=@`*`license-server-name.com`* or `RATIONAL_LICENSE_FILE=@`*`license-server-ip-address`*.

Viewing license information for installed packages

You can review license information for your installed packages, including license types and expiration dates, from IBM® Installation Manager.

1. Start IBM® Installation Manager.
2. On the main page, click **Manage Licenses**.
   The package vendor, current license types, and expiration dates are displayed for each installed package.

Note: See this support article for license upgrade information: https://www.ibm.com/support/pages/node/87467

Starting Rational® Functional TesterHCL OneTest™ UI from the command line

You can start Rational® Functional TesterHCL OneTest™ UI from the desktop environment or a command-line interface.

- For Windows®: To start Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration from the command line, type:

```
<FT installation directory>\eclipse.exe -product com.ibm.rational.rft.product.ide
```

- For Windows®: To start Rational® Functional TesterHCL OneTest™ UI Microsoft Visual Studio .NET Integration, from the command line, type:

```
"<Visual Studio installation directory>\Common7\IDE\devenv.exe"
```

- For Linux®: To start Rational® Functional TesterHCL OneTest™ UI from the command line, close the terminal from where you installed Rational® Functional TesterHCL OneTest™ UI, start a new terminal and type:

```
<product installation directory>\ft_starter
```

  This is required because product environment variables are set during installation. These environment variables are not available to the shell that started the installation process, and therefore it is recommended that you use a new terminal.

  Note:
  - On Ubuntu, you must ensure that the environment variables that are set while installing the products are retained when you open Rational® Functional TesterHCL OneTest™ UI and the application-under-test.
  - When you launch Rational® Functional TesterHCL OneTest™ UI on Linux, a terminal window opens. You must not close the terminal window manually when the application is in use. When you quit the application, the terminal window closes automatically.

Software configuration management

You can integrateRational® Functional TesterHCL OneTest™ UI with the software configuration management systems to maintain an auditable and repeatable history of your organization's test assets.

Benefits of software configuration management

Every time a test script is created, whether through recording or coding, a file is generated that contains code. When created, developed, or edited, that code is a valuable test asset.

A team environment presents the risk of losing functioning code or breaking test scripts by overwriting files. A software configuration management system offers a way to overcome this risk. Every time a file changes, a new version is created and the original file is preserved.

A test script is a collection of files. The complexity of treating several files as a single entity is hidden because all actions in the product user interface are performed on the script. You do not see some of the related files anywhere in the user interface. In addition, some software configuration management operations, such as merging, are very complex. There is built-in logic to determine the order in which files are merged, and then different utilities are employed as needed to complete the merge.

For the team that is new to software configuration management, all of the essential features for versioning test scripts are available through the Rational® Functional TesterHCL OneTest™ UI interface. This integration simplifies the use and adoption of software configuration management.

Note: Use a software configuration management like IBM® Engineering Workflow Management if multiple users must access functional test assets in a test team environment.

Software configuration management products

IBM® Engineering Workflow Management

You can use Jazz source control to manage source code, documents, and other artifacts that you want to place under version control and share with a team. Jazz source control is closely integrated with the other application development lifecycle tools included in Rational Team Concert.

- You can create a project in your workspace, share the project to place the project under Jazz source control.
- Check-in your changes to the repository workspace.
- Deliver the changes to the stream from the repository workspace so that the changes are available to all members of the team.
- You can accept a team invitation, or create a new repository workspace from one of the streams of the team.

For more information, see Testing with IBM Engineering Workflow Management on page 514 and Testing with IBM Engineering Workflow Management on page 355.

EGit

You can store your test assets in the remote Git repository and use EGit for version control operations. EGit is an Eclipse plugin for the Git version control system.

For more information, see Testing with EGit on page 336.

Integrations in UI Test perspective

In this section, you will learn about the supported integrations for the Web UI Test perspective.

Integration plugin compatibility matrix

You can find information about the versions of the integration plugins that are required to integrate Jenkins, Ant, HCL® Launch, and UrbanCode™ Deploy with Rational® Functional TesterHCL OneTest™ UI.

The following table lists the versions of the integration plugins for the UI Test persepctive.

Note: You must download the required version of the integration plugin from the IBM WebSphere, Liberty & DevOps Community portalHCL® License & Delivery portal based on the existing version of Rational®

Functional TesterHCL OneTest™ UI. You can then integrate Jenkins, Ant, HCL® Launch, and UrbanCode™ Deploy with Rational® Functional TesterHCL OneTest™ UI.

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version
10.1.0	RFT-WebUI-Ant-6.0	RFT-WebUI-Jenkins-8.0	RFT-WebUI-UCD-6.0
10.1.1	RFT-WebUI-Ant-6.0	RFT-WebUI-Jenkins-9.0	RFT-WebUI-UCD-7.0
10.1.2	RFT-WebUI-Ant-6.0	RFT-WebUI-Jenkins-9.0	RFT-WebUI-UCD-7.0
10.1.3	RFT-WebUI-Ant-6.0	RFT-WebUI-Jenkins-9.0	RFT-WebUI-UCD-7.0
10.2.0	RFT-WebUI-Ant-6.0	RFT-WebUI-Jenkins-9.0	RFT-WebUI-UCD-8.0
10.2.1	RFT-WebUI-Ant-7.0	RFT-WebUI-Jenkins-10.0	RFT-WebUI-UCD-8.0
10.2.2	RFT-WebUI-Ant-8.0	RFT-WebUI-Jenkins-11.0	RFT-WebUI-UCD-8.0
10.2.3	RFT-WebUI-Ant-8.0	RFT-WebUI-Jenkins-11.0	RFT-WebUI-UCD-10.1
10.5.0	RFT-Ant-UITest-8.0	RFT-Jenkins-UITest-11.1	RFT-UCD-UITest-11.0
10.5.1	RFT-Ant-UITest-8.0	RFT-Jenkins-UITest-11.2	RFT-UCD-UITest-11.1
10.5.2	RFT-Ant-UITest-8.0	RFT-Jenkins-UITest-11.3	RFT-UCD-UITest-11.1
10.5.3	RFT-Ant-UITest-8.0	RFT-Jenkins-UITest-11.4	RFT-UCD-UITest-11.1

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version	HCL® Launch plugin version
10.1.0	HOT-UI-Web-UI-Ant-6.0	HOT-UI-WebUI-Jenkins-8.0	HOT-UI-Web-UI-UCD-6.0	HOT-UI-Web-UI-LAUNCH-6.0

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version	HCL® Launch plugin version
10.1.1	HOT-UI-Web-UI-Ant-6.0	HOT-UI-WebUI-Jenkins-9.0	HOT-UI-Web-UI-UCD-7.0	HOT-UI-Web-UI-LAUNCH-7.0
10.1.2	HOT-UI-Web-UI-Ant-6.0	HOT-UI-WebUI-Jenkins-9.0	HOT-UI-Web-UI-UCD-7.0	HOT-UI-Web-UI-LAUNCH-7.0
10.1.3	HOT-UI-Web-UI-Ant-6.0	HOT-UI-WebUI-Jenkins-9.0	HOT-UI-Web-UI-UCD-7.0	HOT-UI-Web-UI-LAUNCH-7.0
10.2.0	HOT-UI-Web-UI-Ant-6.0	HOT-UI-WebUI-Jenkins-9.0	HOT-UI-Web-UI-UCD-8.0	HOT-UI-Web-UI-LAUNCH-8.0
10.2.1	HOT-UI-Web-UI-Ant-7.0	HOT-UI-WebUI-Jenkins-10.0	HOT-UI-Web-UI-UCD-8.0	HOT-UI-Web-UI-LAUNCH-8.0
10.2.2	HOT-UI-Web-UI-Ant-8.0	HOT-UI-WebUI-Jenkins-11.0	HOT-UI-Web-UI-UCD-8.0	HOT-UI-Web-UI-LAUNCH-8.0
10.2.3	HOT-UI-Web-UI-Ant-8.0	HOT-UI-WebUI-Jenkins-11.0	HOT-UI-Web-UI-UCD-10.1	HOT-UI-Web-UI-LAUNCH-10.1
10.5.0	HOT-UI-Ant-UITest-8.0	HOT-UI-Jenkins-UITest-11.1	HOT-UI-UCD-UITest-11.0	HOT-UI-LAUNCH-UITest-11.0
10.5.1	HOT-UI-Ant-UITest-8.0	HOT-UI-Jenkins-UITest-11.2	HOT-UI-UCD-UITest-11.1	HOT-UI-LAUNCH-UITest-11.1
10.5.2	HOT-UI-Ant-UITest-8.0	HOT-UI-Jenkins-UITest-11.3	HOT-UI-UCD-UITest-11.1	HOT-UI-LAUNCH-UITest-11.1
10.5.3	HOT-UI-Ant-UITest-8.0	HOT-UI-Jenkins-UITest-11.4	HOT-UI-UCD-UITest-11.1	HOT-UI-LAUNCH-UITest-11.1

Testing with Ant

You can use `Ant` to run compound tests and Web UI tests from the command-line interface.

Before you begin

You must have completed the following tasks:

- Installed Installation Manager.
- Installed Rational® Functional TesterHCL OneTest™ UI.
- Verified that you have test assets residing within Rational® Functional TesterHCL OneTest™ UI.
- Downloaded the Rational® Functional TesterHCL OneTest™ UI Web UI Ant plugin `RFT-WebUI-Ant-8.08.0` from the IBM WebSphere, Liberty & DevOps Community portal HCL® License & Delivery portal on to the computer where you install the product.
- Added `Ant` to the *PATH* environment variable.

About this task

To run Web UI tests on Mac OS, you must add an environment variable that points to the installation directory of Rational® Functional TesterHCL OneTest™ UI.

For example, `export TEST_WORKBENCH_HOME=/opt/IBM/SDPexport TEST_WORKBENCH_HOME=/opt/HCL/HCLOneTest`.

Note: For Windows™ and Linux®, the environment variable is set when you install the product.

1. Extract the following files from the downloaded ant plugin:
   - `RFT-Ant-UITest-`*x*`.0.jar` `HOT-UI-Ant-UITest-`*x*`.0.jar`

     Where, *x* is the version number of the `Ant` plugin.
   - `RFT-Ant-UITest.xmlHOT-UI-Ant-UITest.xml`
   - `README.txt`
2. Open the `RFT-Ant-UITest.xmlHOT-UI-Ant-UITest.xml` file and provide required parameter values.

   You must provide the values for the following required parameters:
   - **name**
   - **workspace**
   - **project**
   - **suite**

   If you include these required parameters in a configuration file and use the **Configfile** parameter to specify the complete file path, then these parameters are not required.

   Remember: You must consider the following requirements:

- Enter the parameter values within the double quotation marks.
- Ensure that the special characters in the parameter values do not break the validation of the XML file. For example, you must enter the & character as &.

For example,

```
<webui name="test1" workspace="C:\workspace" projectname="TestProject" suite="Tests/test1.testsuite"
results="Results/test1_on_anttask" />
```

Note: You can add an additional <webui> task and provide the details for each test to run multiple tests simultaneously.

The following table explains each parameter in detail.

Parameter	Description
Required	
name	The name of the test for the particular test product.
workspace	The complete path to the Eclipse workspace.
projectname	The path, including the file name of the project relative to the workspace.
suite	The path, including the file name of the test to run relative to the project. A test can be a Web UI test, compound test, or an Accelerated Functional Test. **Note:** You must provide the file name along with the file extension if you are using an Accelerated Functional Test suite.
Optional	
configfile	The complete path to a file that contains the parameters for a test run.
exportReport	The option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. The command syntax is as follows: exportReport="type=<reporttype>;format=<file type1,file type2,file type3>;folder<destination folder path>;filename=<name of the exported file>" For example, to export the report to only the pdf format, you can use exportReport "type=unified;format=pdf;folder=Exportedreport102;filename=testreport

Parameter	Description
	If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use exportReport "type=unified;format=pdf,xml,html;folder=Exportedreport102;filename=testreport The report in different file formats use the same file name that is specified in the command.
exportstatreportlist	A comma-separated list of absolute paths to custom report format files (.view files) to use when exporting statistical report data with the **exportstats** option.
exportstats	The complete path to a directory that can be used to store exported statistical report data.
exportstatsformat	The option to specify a format for the result that you want to export along with the **exportstats** option. You must use at least one of the following parameters with the **exportstatsformat** option: ◦ simple.csv ◦ full.csv ◦ simple.json ◦ full.json ◦ csv ◦ json For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*simple.json*" You can add multiple arguments separated by a comma. For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*simple.json*, *full.csv*" When you want to export both simple and full type of test results in a json or csv format, you can specify *json* or *csv* as the arguments in the command. When the test run completes, the test result exports to simple.json and full.json files. For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*json*"
exportstatshtml	The complete path to a directory that can be used to export web analytic results. The results are exported in the specified directory. Analyze the results on a web browser without using Rational® Functional TesterHCL OneTest™ UI.

Parameter	Description
imports	When you want to run Web UI tests that are in a source control system such as Git from a computer that runs the desktop product, you can clone the project resources in the remote repository to your computer. You can use an empty workspace folder on your computer to import the UI Test project resources and then run the tests. The desktop product is enabled to run the Web UI tests without the need of the workspace in the cloned repository or your existing workspace. You must use the workspace argument to precede the imports argument. **Note:** You can use this argument in the following scenarios: ◦ You do not want to use your existing workspace. ◦ You do not want to use the workspace cloned from a remote repository. To run UI tests contained in UI Test projects that are in a remote repository, you must perform the following steps: a. Clone the remote repository that contains the UI Test project to your computer. b. Create an empty workspace on your computer. c. Add the following command to the `RFT-Ant-UITest.xml`HOT-UI-Ant-`UITest.xml` file: name="<test_name>" workspace="<path_of_empty_workspace>" project="<project_name>" imports="<path_to_cloned_project_folder>" suite="<test_name>" For example, name="test1" workspace="C:\workspace" project="UIProject1" imports="d:\work\UIProject1" suite="Test1"
imsharedloc	The complete path to IBMIMSharedHCLIMShared location, if it is not at the default location.
labels	The option to add labels to the test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, **labels** =*"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use **labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset.

Parameter	Description
	Note: ◦ When you run tests by using the double quotation marks ("") for the **labels** parameter, then the labels in the test result do not include the double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **configfile** parameter. ◦ When you use the **configfile** parameter to run tests, then labels provided in the configuration file take precedence over the labels provided in the `RFT-Ant-UITest.xmlHOT-UI-Ant-UITest.xml` file.
overwrite	Determines whether a result file with the same name is overwritten. The default value is `false`, which means the result file cannot be overwritten and a new result file is created.

protocolinput	The option to run a Web UI test in parallel on different browsers.
	`protocolinput="all.available.targets.in.parallel=all"`
	`protocolinput="all.available.targets.in.parallel=chrome,ff,ie"`
	Note: If you use the `protocolinput` argument, you must not use the equivalent vmargs arguments:
	`vmargs="-Dall.available.targets.in.parallel=all"`
	`vmargs "-Dall.available.targets.in.parallel=browser1,browser2,browser3"`

publish	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **publish** parameter: ◦ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _-teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _-teamspace_alias* You must consider the following points while providing the project name:

Parameter	Description

	▪ If the project name is not specified, then the value of the **Project** parameter is used.
	▪ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias==***name_of_the_teamspace_alias* options.
	For example:
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1"`
	Where:
	▪ https://localhost:5443 is the URL of the server.
	▪ test is the name of the project.
	▪ ts1 is the name of the team space.
	▪ While providing the name of the team space or team space alias, you must replace the ampersand (&) character with & as shown in the preceding example.
	▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`.
	For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*.
	Where, %20 is the hexadecimal value of the space character.
	◦ To avoid publishing of reports, use **no**.
	You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
	For example:
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="no"`
	If you do not use the **configfile** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences

Parameter	Description
	(**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**). The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions: ◦ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**. ◦ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
publish_for	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **publish_for** parameter along with the **publish** parameter. The following are the available options that you can use for the **publish_for** parameter: ◦ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict. ◦ **PASS** - You can use this option to publish test results for the tests that have passed. ◦ **FAIL** - You can use this option to publish test results for the tests that have failed. ◦ **ERROR** - You can use this option to publish test results for the tests that included errors. ◦ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests. You can add multiple parameters separated by a comma. For example: `name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1" publish_for="FAIL,ERROR"`
publishreports	The option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **publishreports** are as follows:

Parameter	Description

	◦ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455.
	◦ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.
	◦ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
	
	You must use the **publishreports** parameter along with the **publish** parameter.
	
	For example:
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1" publishreports="STATS, TESTLOG"`
	
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	
	For example,
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1" publishreports="!TESTLOG"`
	
	All the reports except the TESTLOG report is published to Rational® Test Automation Server-HCL OneTest™ Server after executing the command.

results	The name of the results file. The default result file is the test name with a time stamp appended.
swapdatasets	For a test, the default value is the dataset specified in the test editor. You must use the **swapdatasets** option to replace dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example: `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`. You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a semicolon (;).

Parameter	Description
usercomments	The option to add text within the double quotation mark (“”) to display it in the **User Comments** row of the report. **Note:** ◦ When you run tests by using the double quotation marks ("") for the **usercomments** parameter, then the **User Comments** row of a report does not contain double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **configfile** parameter.
varfile	The complete path to the XML file that contains the variable name and value pairs.
vmargs	To pass Java™ virtual machine arguments.

3. Open a command prompt and navigate to the directory where you downloaded the `Ant` plugin.

   Note:

   You must close Rational® Functional TesterHCL OneTest™ UI before you run the test.
4. Enter ant -f RFT-Ant-UITest.xmlant -f HOT-UI-Ant-UITest.xml to run the test.

Results

You have run the test by using the `Ant` plugin.

What to do next

You can view that the `Ant` execution output is logged into the `logfile.txt` file, and a test log is created in a temp directory called `RFT-Ant-UITest-x.0` `HOT-UI-Ant-UITest-x.0`.

Testing with Azure DevOps for UI tests

When you use Azure DevOps for continuous integration and continuous deployment of your application, you can create tests for your application in Rational® Functional TesterHCL OneTest™ UI and run those tests in Azure DevOps pipelines. You can integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension that is available in the **Visual Studio Marketplace** portal.

Prerequisites

Before you integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI, you must have completed certain tasks. See Prerequisites for Azure DevOps Integration on page 318.

Overview

You can use the *IBM Rational Test WorkbenchHCL OneTest Studio* extension that enables you to select any type of test created in Rational® Functional TesterHCL OneTest™ UI that you can add to your task for the job in the Azure DevOps pipelines.

Running tests

Click the link to the task information for the type of tests that you want to run from the following types:

- For AFT Suites, Compound Tests, or Web UI tests, see Testing with Azure DevOps for UI tests on page 317.
- For functional tests, see Testing with Azure DevOps for functional tests on page 462.

Prerequisites for Azure DevOps integration with Rational® Functional TesterHCL OneTest™ UI

Before you integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension, you must have completed certain tasks.

- You must have installed Rational® Functional TesterHCL OneTest™ UI on a computer running Windows™ or Linux®.
- You must have created an organization and a project in Azure DevOps for running jobs in Azure DevOps pipelines. For more information refer to Creating an organization.

You can now follow the tasks listed in the task flow table to integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI. See Task flow for integrating Azure DevOps on page 318.

Task flow for integrating Azure DevOps with Rational® Functional TesterHCL OneTest™ UI

The table shows the task flow for integrating Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension. You must perform these tasks in sequence as listed in the following table. The table also provides you the links to the information about the tasks.

	Tasks	More information
1	Create any or all of the following types of tests in Rational® Functional TesterHCL OneTest™ UI to test your application: • Accelerated Functional Testing (AFT) Suites • Web UI tests • Compound tests • Traditional functional tests	Testing in the UI Test perspective on page 532

	Tasks	More information
2	Access the **Visual Studio Marketplace** portal and search for the latest version of the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.	Visual Studio Marketplace
3	Install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.	Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension on page 319
4	Run tests in an Azure DevOps pipeline.	Running tests in an Azure DevOps Pipeline on page 320

Related information

Testing with Azure DevOps for UI tests on page 317

Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension

You must install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization before you use the extension to run tests for your application in an Azure DevOps pipeline. The *IBM Rational Test WorkbenchHCL OneTest Studio* extension supports running of tests created in Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have access to the **Visual Studio Marketplace** portal.

About this task

After you install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension from the **Visual Studio Marketplace** portal in your Azure DevOps organization, you can use the extension to run tests for your application in an Azure DevOps pipeline.

1. Log in to the **Visual Studio Marketplace** portal, if you are not already logged in.
2. Click the **Azure DevOps** tab.
3. Search for the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
4. Click the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
5. Click **Get it free**.

   Result

   The **Visual Studio Marketplace** portal for the *IBM Rational Test WorkbenchHCL OneTest Studio* extension is displayed.
6. Select the organization where you want to run your test from the **Select an Azure DevOps Organization** list.
7. Click **Install**.

   Result

The installation is completed.

8. Click **Proceed to organization**.
   Result

   The **Organization** page in Azure DevOps is displayed.

9. Click **Organization settings > Extensions**.
   Result

   The *IBM Rational Test WorkbenchHCL OneTest Studio* extension is displayed as an installed extension.

Results

You installed the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization.

What to do next

You can add tests that you created in Rational® Functional TesterHCL OneTest™ UI to your task, and then run the tests in an Azure DevOps pipeline. See Running UI tests in an Azure DevOps Pipeline on page 320.

Running UI tests in an Azure DevOps Pipeline

After you create the tests in IBM® Rational® Functional TesterHCL OneTest™ UI for the application that you are testing, and after you install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your organization, you can run the tests in Azure DevOps pipelines.

Before you begin

You must have completed the following tasks:

- Installed the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your organization. See Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension on page 319.
- Installed an agent in your pipeline. See Azure Pipelines agents.
- Created test cases under test plans if you want to view results of the test runs on the Test Plans dashboard.

About this task

After you add the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization, you can use an existing pipeline or create a new one to add Rational® Functional TesterHCL OneTest™ UI test tasks. You can install an agent or use the one that you installed in your default agent pool. You can add the Rational® Functional TesterHCL OneTest™ UI tests to your task for the agent job, configure the task, and then run the task in the Azure DevOps pipeline.

If you have created test cases under test plans in your Azure DevOps project, you can provide the details of the Azure DevOps URL, test plan, test case, and your personal access token (PAT) while you configure the test job in a pipeline so that you can view the results of the test run on your Test Plan dashboard.

1. Open your **Organization** page in Azure DevOps and perform the following steps:
    a. Click the project you want to use.
    b. Initialize the repository by performing the following steps:
        i. Click **Repos** from the left pane.
        ii. Click **Initialize** from the **Initialize with a README or gitignore** section.

        Note: Select the **Add a README** check box if it is not selected.

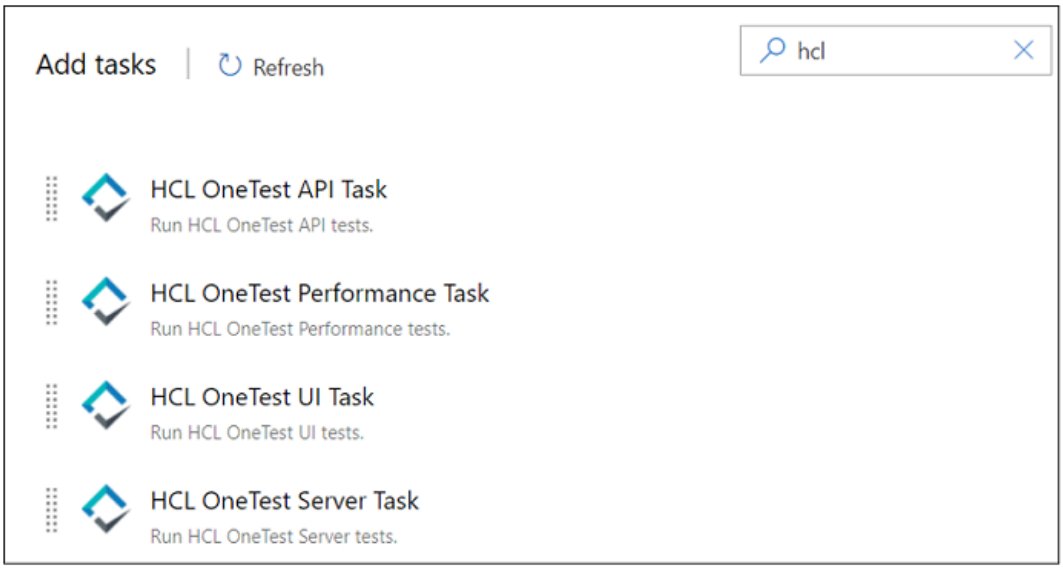
    c. Click **Pipelines** from the left pane.
    d. Click **Create Pipeline**.
    e. Click **Use the classic editor** to create a pipeline without YAML.
    f. Verify the project, repository, and branch for manual and scheduled builds, and then click **Continue**.
    g. Click **Empty job**.
2. Select **Pipeline** and complete the following steps:
    a. Change the name for the build pipeline if required.
    b. Select the **Agent pool** for your build pipeline.

        You can use the agent from the default agent pool or use the one you have installed.
    c. Select the **Agent Specification** for the agent if required.
3. Add a task to the agent job by completing the following steps:
    a. Click the **Add Task** icon for the agent job.
    Result

    The **Add tasks** pane is displayed.
    b. Search for the IBMHCL tasks defined in the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
    Result

    The tasks that you can select are displayed.





Depending on the type of test that you have created in Rational® Functional TesterHCL OneTest™ UI, you can select the type of task.

You must use the following table to identify the task you must select:

Type of test	Task to select
▪ AFT suites ▪ Web UI tests ▪ Compound tests	IBM® Rational® Functional TesterHCL OneTest™ UI Task

c. Select the **Rational Functional Tester TaskHCL OneTest UI** option, and then click **Add** to add the task to the agent job.

Result

The selected task is added to the agent job and it is displayed with a warning that some settings require attention. You must configure the settings mentioned in Step 4 on page 323.

You can also remove the tasks that are not required in your job. Select the tasks in the list that you want to remove. You can then right-click the tasks, and click **Remove selected task(s)** to remove them.

4. Configure the settings by performing the following steps:

   a. Select the task version from the list if required.

   b. Follow the action for the Web UI task by referring to the following table:



Note: All the required fields are marked with an asterisk (*) in the UI.



Note:

You must provide the values for the following required fields:

- **Workspace Location**
- **Project Name**
- **Test Suite Name**

If you include these required parameters in a configuration file and use the **Configuration File** field to specify the complete file path, then these values are not required.

Field	Description	Action
Display name	Displays the name of the selected task.	Enter the name of the task.
Testcase Type	The type of test to execute.	Select **Web UI** from the **Testcase Type** list.
Import	The complete path to the project folder that is cloned from a source control system or remote repository.	Enter the complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets.
IMShared Path	The path to the IMShared folder on your local computer.	Enter the complete path to the location of the IBMIMSharedHCLIMShared folder if it is not in the default path. For example, `D:\IBM\IBMIMSharedD:\HCL\HCLIMShared`

Field	Description	Action
		Note: If you do not enter any value, the default path of the IBMIMSharedHCLIMShared folder is considered for test run. The default path in the different operating systems is as follows: ▪ Linux: `/opt/IBM/IBMIMShared/opt/HCL/HCLIMShared` ▪ Mac: `/Application/IBM/IBMIMShared/Application/HCL/HCLIMShared` ▪ Windows: `C:\Program Files\IBM\IBMIMSharedC:\Program Files\HCL\HCLIMShared`
Workspace Location	The complete path to the Eclipse workspace.	Enter the complete path of the Eclipse workspace.
Project Name	The name of the project containing the test.	Enter the name of the project containing the test.
Test Suite Name	The name of a test within the project to use. A test can be a Web UI test, compound test, or AFT suites.	Enter the name of the test that you want to run.
VM Arguments	Java™ virtual machine arguments to pass in.	Enter the Java™ virtual machine arguments. **Note:** You can add multiple virtual machine arguments files separated by a `comma`.
Labels	The option to add labels to the test results when the test run is complete.	Enter the labels to be added to the test results when the test run is complete. You

Field	Description	Action
		can add multiple labels to a test result separated by a comma. For example, *"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:** ▪ When you run tests by using the double quotation marks ("") for the **Labels** field, then the labels in the test result do not include the double quotation marks. ▪ To work around this problem, you must create a command-line config file, and then run the test by using the **Configuration File** field. ▪ When you use the **Configuration File** field to run tests, then labels provided in the configuration file

Field	Description	Action
		take precedence over the labels provided in the **Labels** field.
Var File	The complete path to the XML file that contains the variable name and value pairs.	Enter the complete path to the location of the variable file.
Dataset Override	For a test, the default value is the dataset specified in the test editor.	Use the `Dataset Override` option to replace dataset values during a test run. If a test is associated with a dataset, you can replace the dataset at run time while initiating the run from the command line. **Note:** You must use the `Dataset Override` option to replace the dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example, `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`. You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a `semicolon (;)`. For example, `/project_name1/ds_path/ds_filename.csv:/project_name1/ds_path/new_ds_filename.csv;`

Field	Description	Action
		`/project_name2/ds_path/ds_filename.csv:/project_name2/ds_path/new_ds_filename.csv`
Configuration File	The complete path to a file that contains the parameters for a test run.	Enter the complete path of the file that contains the parameters for a test run.
Results	The name of the results file. The default name of the result file is the test name with a time stamp appended.	Enter a folder name that is relative to the project to store the test results. For example, `-results folder/resultname`.
Overwrite	Determines whether a results file with the same name is overwritten. The default value `false` indicates that the new results file is created.	Set the value as `true` to overwrite the existing file and retain the same file name.
Export Stats	The complete path to export the report to the format specified in the **Export stats format** field.	Enter the values separated by comma to export those values to the CSV format.
Export stat report list	The option to list the reports that you want to export in place of the default reports, or the reports that are selected under **Preferences**.	Enter the list of report IDs separated by comma.
Export stats html	The option to view and analyze the results on a web browser without using the test workbench.	You can provide the complete path to a directory to export web analytic results. If you run multiple tests, do not provide a value in this field.
User Comments	The option to add and display your comments in the report.	Enter the text within double quotation mark to display it in the **User Comments** row of the report. **Note:** ▪ When you run tests by using the double quotation marks ("") for the **User Comments** field, then the **User Comments** row of a report does not contain double quotation marks. ▪ To work around this problem, you must create a

Field	Description	Action
		 command-line config file, and then run the test by using the **Configuration File** field.
Protocol Input	The option to run a Web UI test in parallel on different browsers.	Enter the browsers on which you want to run the Web UI test in parallel.
Export Report	The option to export the unified report to other file formats such as PDF, XML, and HTML.	Enter the following details in the text field: *type=<report type>;format=<file type>;folder=<destination folder path>;filename=<name of the exported file>*
Export stats format	The option to specify a format for the report that you want to export.	Specify a format for the report that you want to export. You must use the **Export stats** parameter along with the **Export stats format** parameter. You must use at least one of the following formats: ▪ *simple.csv* ▪ *full.csv* ▪ *simple.json* ▪ *full.json* ▪ *csv* ▪ *json* For example, **Export stats** = *<local_dir_path>* **Export stats format** = *json* You can add multiple formats for the report separated by a comma (,). If you want to export both the simple and full reports in a *json* or *csv* format, you can specify *json* or *csv* as the format in the field.

Field	Description	Action
		The reports are saved to the location specified in the **Export stats** field.
publish	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server.	Specify the server URL and project name to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **publish** parameter: ▪ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias* You must consider the following points while providing the values: ▪ If the project name is not specified, then the value of the **Project Name** parameter is used. ▪ If you have a project with the same name in different

Field	Description	Action

		team spaces, then you can append either the **&team-space.name**=*name_of_-the _teamspace* or **&team-space.alias=**=*name_of_-the_teamspace_alias* options.
		
		For example:
		
		`Workspace Location = C:/Users/IBM/rationalsdp/workspace1`
		`Project Name = proj1`
		`Test Suite Name = Tests/testHttp.testsuite`
		`publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
		
		Where:
		
		▪ https://local-host:5443 is the URL of the server.
		
		▪ test is the name of the project.
		
		▪ ts1 is the name of the team space.
		
		▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_-character>`.
		
		For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*.

Field	Description	Action

		Where, %20 is the hexa-decimal value of the space character.
		▪ You must not include the double quotation marks ("") while providing the value for the **Publish** field.
		▪ To avoid publishing of reports, use **no**.
		You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
		For example:
		`Workspace Location = C:/Users/IBM/rationalsdp/workspace1`
		`Project Name = proj1`
		`Test Suite Name = Tests/testHttp.testsuite`
		`publish = no`
		If you do not use the **Config File** field to run the tests, then the values provided in the **Publish** field always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
		The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:
		▪ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in

Field	Description	Action
		preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**. ▪ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
publish_for	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests.	Specify the option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **publish_for** parameter along with the **publish** parameter. The following are the available options that you can use for the **publish_for** parameter: ▪ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict. ▪ **PASS** - You can use this option to publish test results for the tests that have passed. ▪ **FAIL** - You can use this option to publish test results for the tests that have failed. ▪ **ERROR** - You can use this option to publish test results for the tests that included errors. ▪ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.

Field	Description	Action

		You can add multiple parameters separated by a comma.
		
		For example:
		`Workspace Location = C:/Users/IBM/rationalsdp/workspace1`
		`Project Name = proj1`
		`Test Suite Name = Tests/testHttp.testsuite`
		`publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
		`publish_for = FAIL,ERROR`

publishreports	The option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server.	Specify the option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **publishreports** are as follows:
		▪ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455.
		▪ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.
		▪ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
		
		You must use the **publishreports** parameter along with the **publish** parameter.
		
		For example:
		`Workspace Location = C:/Users/IBM/rationalsdp/workspace1`
		`Project Name = proj1`
		`Test Suite Name = Tests/testHttp.testsuite`

Field	Description	Action
		``` publish = https://localhost:5443#project.name =test&teamspace.name=ts1 publishreports = STATS, TESTLOG ``` The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI. You can prefix the value with "!" to publish the reports except for the specified one in the test script. For example, ``` Workspace Location = C:/Users/IBM/rationalsdp/workspace1 Project Name = proj1 Test Suite Name = Tests/testHttp.testsuite publish = https://localhost:5443#project.name =test&teamspace.name=ts1 publishreports = !TESTLOG ``` All the reports except the TESTLOG report is published to Rational® Test Automation ServerHCL OneTest™ Server after executing the command.
Azure DevOps Project URL	The URL of the test project in the organization on the Azure DevOps server. **Note:** You must enter the details for this option if you have created test cases under Test plans in your Azure project and want to view the test results on the Test Plans Dashboard.	Enter the URL of the test project in the organization on the Azure DevOps server in the following format: https://<host>/<orgname>/<projectName> You must use this option with the following options: ▪ Azure DevOps PAT ▪ Test Plan Name ▪ Test Case Name
Azure DevOps PAT	Your personal access token (PAT) of the Azure server where the test project that	Enter your personal access token.

Field	Description	Action
	contains the test plan is hosted. You can also add your token to a secret variable and specify the variable name in the following form: `$(variable_name)` **Note:** The token must have read and write access.	You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Test Plan Name ▪ Test Case Name
Test Plan Name	The name of the test plan in the Azure server.	Enter the name of the test plan. You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Azure DevOps PAT ▪ Test Case Name
Test Case Name	The name of the test case in the Azure server.	Enter the name of the test case. You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Azure DevOps PAT ▪ Test Plan Name

c. Expand **Control Options** and configure the settings for your task if required.

d. Expand **Output Variables** and configure the settings for your task if required.

5. Select the following options:

a. Click **Save** to save the configured settings for the task.

Note: The task is not queued for a run.

You can save the task to a build pipeline and opt to run the build at a later time.

b. Click **Save & queue** to save the configurations and queue the run in the pipeline.
Result

The **Run pipeline** dialog box is displayed.

6. Complete the following steps:

a. Enter a comment for the test in the **Save comment** field.

b. Select the agent that you configured for the test from the **Agent pool** list.

c. Select the agent specification from the **Agent Specification** list for the agent if required.

d. Select the branch from the **Branch/tag** list.

e. Add the variables and demands for the task run from the **Advanced Options** pane if required.

f. Select the **Enable system diagnostics** check box for a detailed log view.

g. Click **Save and run**.
Result

The **pipeline summary** page displays the progress of the job run.

Results

You have run the tests for the application you are testing, in the Azure DevOps pipeline.

What to do next

- You can open the job to view the task logs from the **pipeline summary** page.
- You can click **Test Plans** to view the test result if you specified the values for **Azure DevOps Project URL**, **Azure DevOps PAT**, **Test Plan Name**, and **Test Case Name**.
- You must click the task to open the **Task** page to view the test results.
- You can access the report URLs to view the test execution information at any point of time. The report URLs are the IBM® Rational® Test Automation ServerHCL OneTest™ Server URLs where the reports are stored.

  In Rational® Functional TesterHCL OneTest™ UI, if the Rational® Test Automation ServerHCL OneTest™ Server URL is configured in **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server** and **Publish result after execution** is set as **Always** in **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server > Results**, then the logs in the **Task** page also displays the names of the published report along with its corresponding URLs.

Related information

Configuration of test runs from the command line on page 1381

Testing with EGit

You can store your test assets in the remote Git repository and use EGit for version control operations. EGit is an Eclipse plugin for the Git version control system.

When you install Rational® Functional TesterHCL OneTest™ UI, EGit is part of the product. To avoid the clutter, EGit is a separate Eclipse perspective in Rational® Functional TesterHCL OneTest™ UI. For more information about EGit, refer to EGit User Guide documentation.

You can click **Windows > Perspective > Open Perspective > Other > Git** to open the Git perspective within Rational® Functional TesterHCL OneTest™ UI.

Refer to the following topics to learn more about EGit integration with Rational® Functional TesterHCL OneTest™ UI.

Importing test assets from the remote Git repository

You can import the test assets from the remote Git repository to Rational® Functional TesterHCL OneTest™ UI and run it to test the performance of the application.

Before you begin

You must have completed the following tasks:

- Stored test assets of an application under test in the remote Git repository.
- Cloned the remote Git repository. For more information refer to the EGit User Guide documentation.

1. Open Rational® Functional TesterHCL OneTest™ UI.
   Result
   The default perspective is the **Performance Test**.
2. Click **File > Import > Git**, and then click the expand icon .
3. Select **Projects from Git**, and then click **Next**.
4. Select the **Existing local repository** option, and then click **Next**.
   Result
   The list of all cloned remote Git repositories is displayed.
5. Select a remote Git repository from the list that has test assets, and then click **Next**.
6. Select one of the following options and perform the steps described in the table based on your requirement:

Options	Descriptions	Steps
Import existing Eclipse projects	Select this option to choose projects that you want to import from the remote Git	Perform the following steps:

Options	Descriptions	Steps
	repository to Rational® Functional Tester-HCL OneTest™ UI. **Note:** If your project folder contains the `.project` file, only then the wizard displays the project that is available for the selection.	a. Select **Import existing Eclipse projects**, and then click **Next**. b. Select the checkbox to choose the projects from the list. c. Click **Finish** to import the projects. The imported project is displayed in the **Test Navigator** pane.
Import as general project	Select this option to choose a project that does not have a `.project` file in your project folder. When you select this option, a `.project` file is created within the project folder.	Perform the following steps: a. Click **Import as general project**, and then click **Next**. b. Enter a name for the project in the **Project name** field. c. Click **Finish**.

Results

You have imported test assets from the remote Git repository.

Pulling the changes from the remote Git repository

You must make your local repository up-to-date by pulling the latest changes in the remote Git repository before you committ and push your updated test assets to the Git repository.

Before you begin

You must have completed the following tasks:

- Cloned the remote Git repository. For more information refer to the EGit User Guide documentation.
- Ensured that any of your team members have updated the test assets and committed those files to the remote Git repository.

About this task

When you updated any of the test assets in your project, you might see merge conflicts errors while pulling the changes, if other members are also working on the same project.

1. Open Rational® Functional TesterHCL OneTest™ UI.
   Result
   The default perspective is the **Performance Test**.
2. Go to **Windows > Perspective > Open Perspective > Other**.

Result

The **Open Perspective** window is displayed.

3. Select the **Git** option from the list, and then click **Open**.

   Result

   The **Git perspective** view is displayed.

4. Right-click the remote Git repository from the list, and then select **Pull**.

Results

You have pulled the new changes from the remote Git repository to your existing local Git repository.

What to do next

You can publish your changes to the remote Git repository. See Publishing test assets to the remote Git repository on page 339.

Publishing test assets to the remote Git repository

When you create or update any test assets in Rational® Functional TesterHCL OneTest™ UI, you can publish your changes to the remote Git repository. Therefore, when you publish test assets, other members in the project can use your test assets in their test run, if required.

Before you begin

You must have completed the following tasks:

- Created or updated test assets in Rational® Functional TesterHCL OneTest™ UI.
- Made the local repository up-to-date by pulling the changes from the remote Git repository. See Pulling the changes from the remote Git repository on page 338.

About this task

You can either publish only the updated test asset files or entire project to the remote Git repository.

1. Open Rational® Functional TesterHCL OneTest™ UI.

   Result

   The default perspective is the **Performance Test**.

2. Locate the test assets files or projects that you updated from the **Test Navigator** pane.
3. Right-click the test assets or project, and then click **Team > Commit**.

   Result

   The **Git Staging** view is displayed.

4. Select the updated test asset files from the **Unstaged Changes** pane.
5. Click  or  to select a specific file or all the files from the **Unstaged Changes** pane to the **Staged Changes** pane.
6. Enter a message in the **Commit Message** pane.
7. Click **Commit and Push**.

Result

The **Push Result** dialog box is displayed with the status of the commit to the remote Git repository.

Results

You have published test assets to the remote Git repository.

Techniques for troubleshooting issues in EGit

You might encounter issues while working with EGit in Rational® Functional TesterHCL OneTest™ UI. You can find the information about certain issues and how to resolve them.

Warning: When you initialize a new Git repository for a project in an Eclipse perspective, a `.gitignore` file is created in the project folder. While you commit the contents of a project to the Git repository, the `.gitignore` file ignores the report files. You can specify additional file types in the `.gitignore` file so that those file types are ignored when you commit the project contents to the Git repository.

The following topics provide more information about EGit integration issues, their causes, and the resolutions to fix the issues while you work with the Git perspective in Rational® Functional TesterHCL OneTest™ UI.

Error: Missing library files in the project

After you pull a project in the Git perspective, `Project is missing the required Library` error is displayed in the **Verify Problems** tab. This error occurs due to the `.classpath` file which is specific to a workspace or system. The `.classpath` file needs to be pointed to the newly imported location.

You can perform the following steps to resolve the missing library issue:

1. Go to **Windows > Perspective > Open Perspective > Other**.
2. Select the **Java** perspective.
3. Expand the project from the **Test Navigator** pane.
4. Navigate to **Java Build Path > Libraries**.
5. Delete all the jar files that are missing after you pull the changes.

   Note: The files are marked with a red cross.

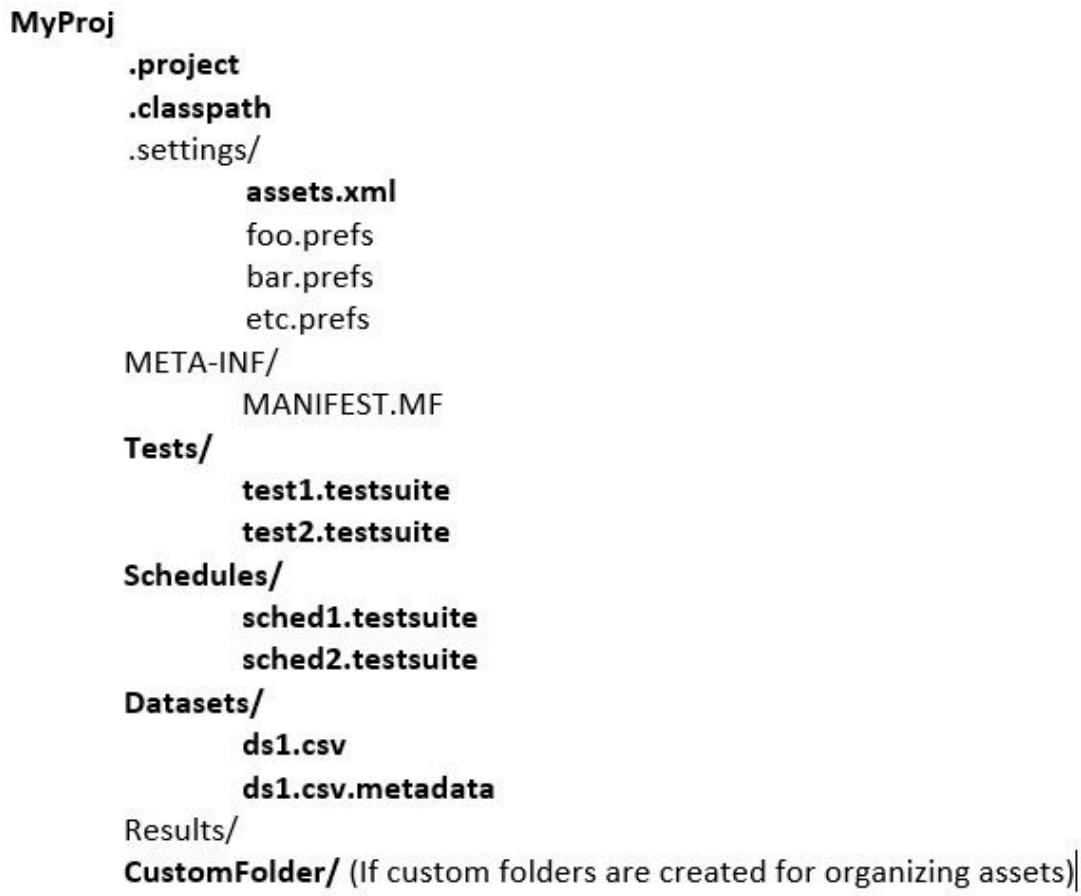
6. Play back the project.

   The required jar files are added to the project.

Git merge conflicts

Web UI test projects contain certain metafiles and test assets in binary form. When you work in a collaborative test creation environment, you might encounter Git merge conflicts while you commit test assets to the remote Git repository.

When you view a project in the File Explorer, the folder structure is displayed as follows:



The metafiles such as `.project`, `.classpath`, `assets.xml`, and the test assets must be committed to the remote Git repository for seamless collaboration. Due to the specialized format of information in the metafiles and test assets being binary files, the Git merges cannot happen automatically. You must resolve the resulting merge conflicts manually, if any.

Best practices to minimize the merge conflicts

You must adhere to the following best practices to minimize the merge conflicts when you commit test assets to the remote Git repository:

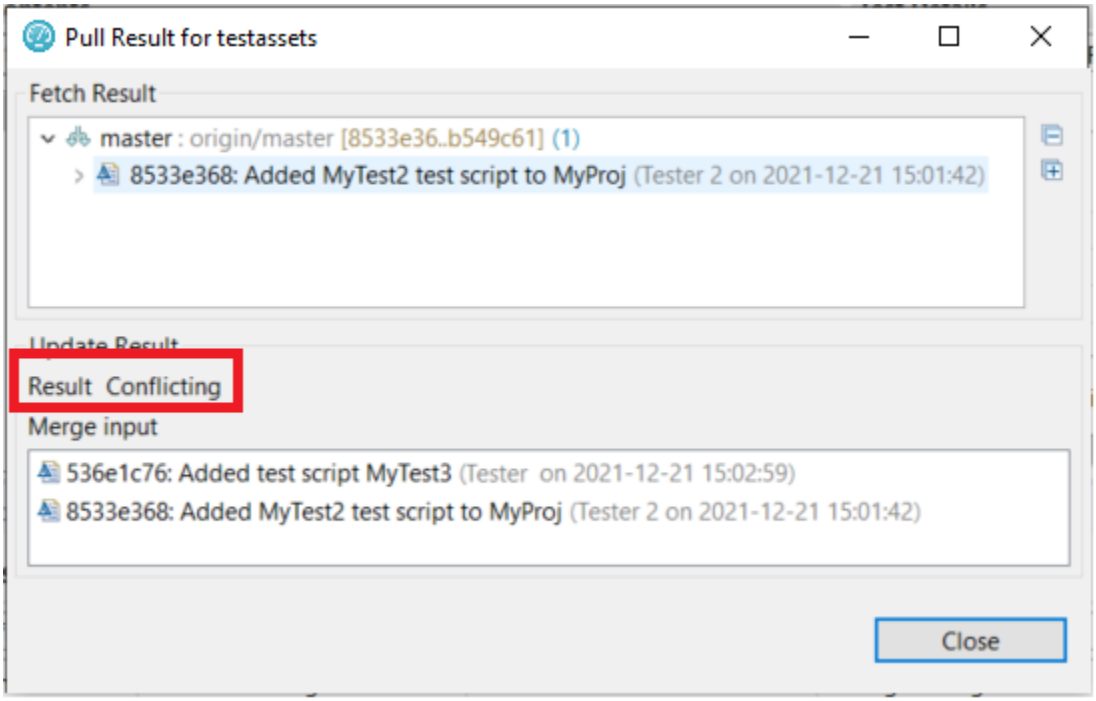
- Refrain from working on the same test assets to the extent, if possible.
- Ensure that the team members coordinate and work closely with other members while making changes on the same test assets. Thereby, if there are any merge conflicts on binary files, those errors can be identified and addressed at the earliest.
- Keep the local Git repository up-to-date with the remote upstream branch by pulling in changes periodically. The local Git branch must be made up-to-date with the master branch by re-basing frequently.
- Use the descriptive messages for your commits, so that other members can refer to those messages to resolve merge conflicts efficiently.

Conflict management: errors in assets.xml

In a collaborative work environment, the merge conflicts in `assets.xml` arise despite best efforts when you pull the changes from the remote upstream.

The project includes a metafile called `assets.xml` that contains a list of all test asset files in a project along with their dependencies in an `XML` format. The `assets.xml` file is automatically generated by Rational® Functional TesterHCL OneTest™ UI from the contents of the project. The `assets.xml` file is refreshed each time you modify the files in the project. These changes in the `assets.xml` file are the most common source of merge conflicts.

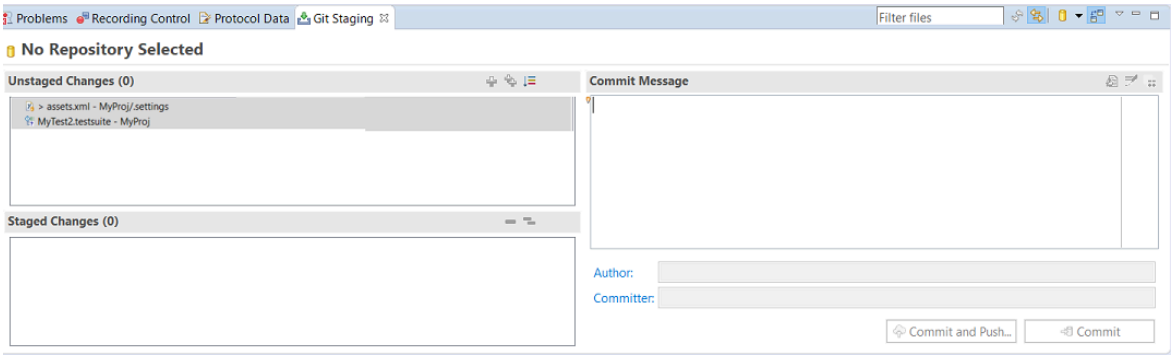
The following image displays the merge conflicts when you attempt to perform a pull request on the local Git repository by using the **Git** perspective:



You can perform the following steps to resolve the merge conflicts in the `assets.xml` file:

1. Click **Close** to dismiss the merge conflict result dialog box.
2. Click **Window > Show View > Other**.
3. Expand **Git**, and then select **Git Staging**.
4. Click **Open**.

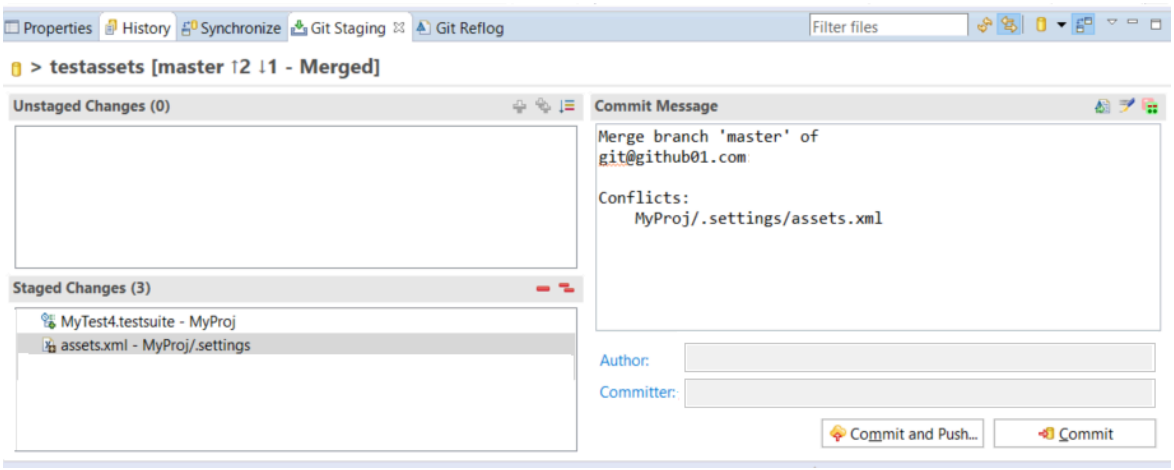
   The conflicting files are displayed in the **Unstaged Changes** pane.



5. Right-click the project from the **Test Navigator** pane, and then click **Refresh**.

   When you refresh the project, the `assets.xml` file is updated to include the local files in the project and also files that were pulled down from the remote upstream.

6. Go to the **Git Staging** view, and then move the files from **Unstaged Changes** to **Staged Changes** by using the **Add selected files to the index** or icon.



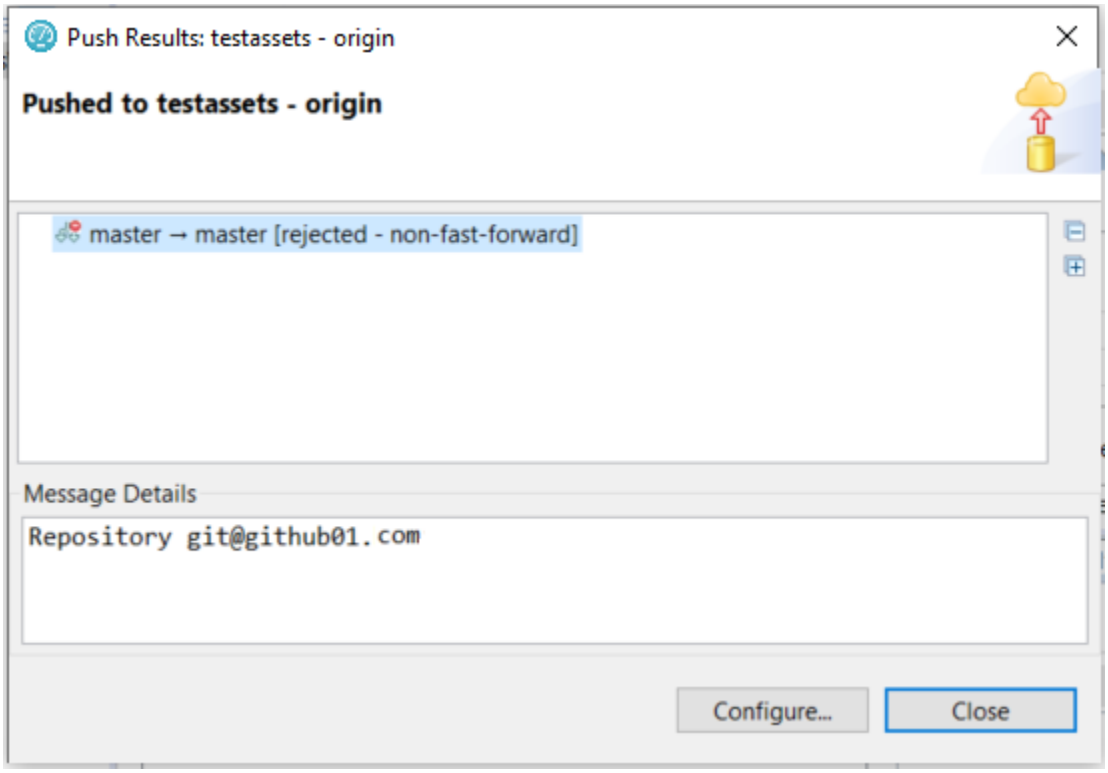
7. Enter a message in the **Commit Message** pane, and then click **Commit and Push**.

   The **Push Result** dialog box is displayed with the status of the commit to the remote Git repository.

Conflict management: errors in test assets

When multiple users edit and commit the same set of test assets, then while you pull or push changes to the remote Git repository, a merge conflict occurs.

Consider two users are working on the same VU Schedule called `MyVUSchedule1.testsuite`. One of the users committed and pushed changes to the remote Git repository. Later, when the other user tries to commit and push changes, the commit fails with the following error message:

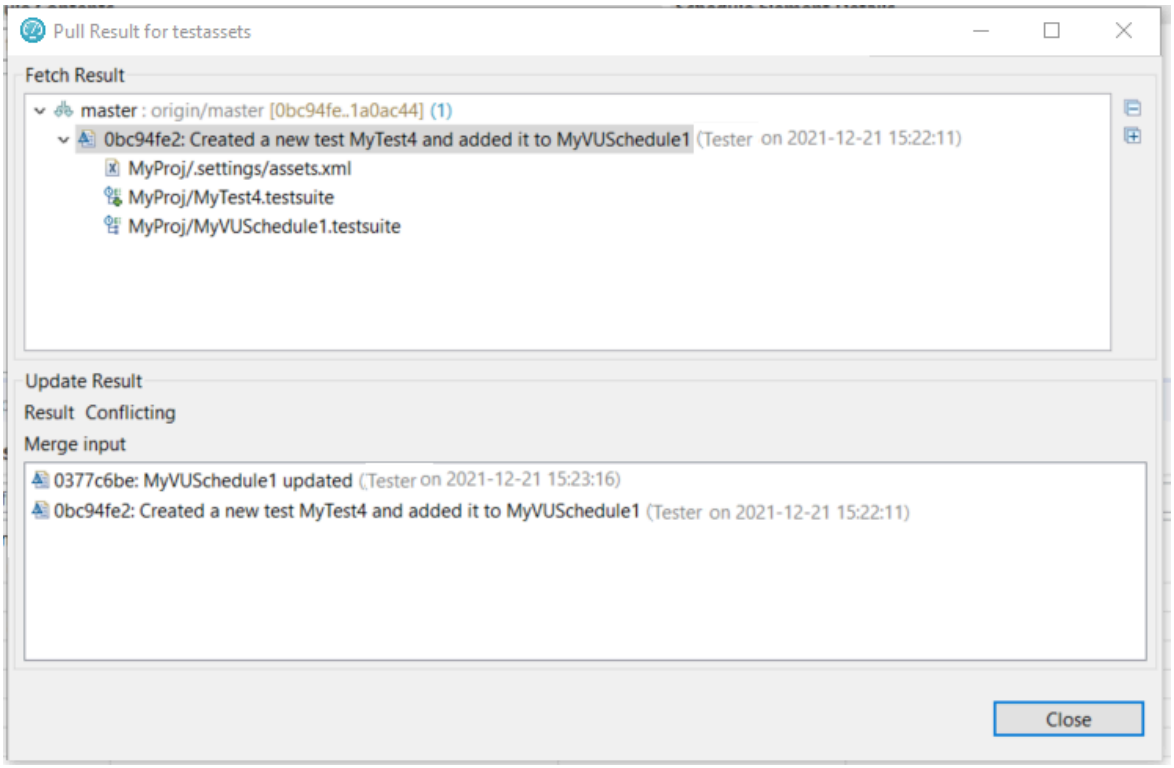


The error occurs because the remote Git repository contains changes pushed by the other user that are not present in the local branch. To push the changes, the changes in the remote Git repository must be pulled down to the local repository.

You can perform the following steps to resolve the merge conflicts in test assets:

1. Go to the **Git perspective** view, and then right-click the remote Git repository from the list.
2. Select **Pull** from the list to make your local Git repository up-to-date.

   The **Pull Request** window is displayed.
3. Expand the commits fetched from the remote Git repository, and then expand each commit to view the list of changed files in them.



Note: You must make a note of the test assets listed in the **Pull Request** window that were modified locally.

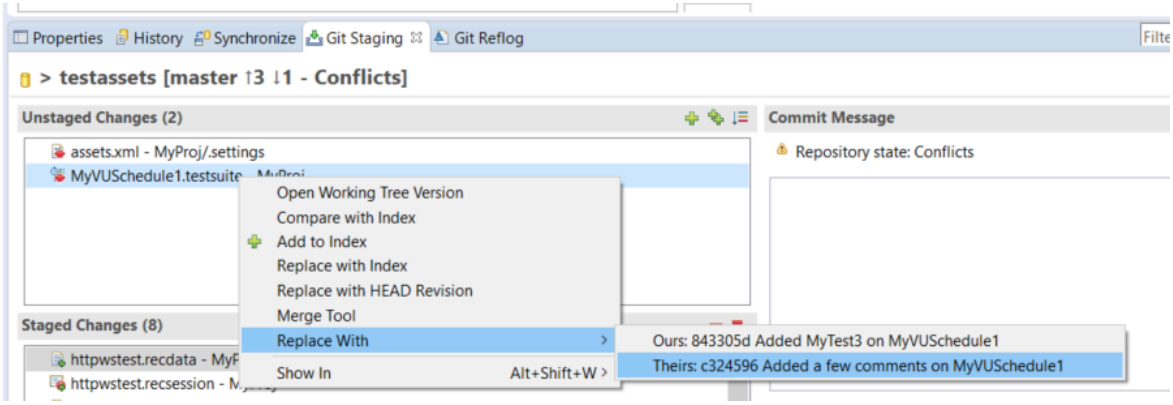
4. Examine the **Git Staging** view to find the conflicting or unmerged files.

Note: The conflicting files are marked with a red icon.

5. Right-click the conflicted test assets, and then select the **Replace With** option.
6. Select one of the following options based on your requirement:
    - **Theirs:commitID commitmessage**
    - **Ours:commitID commitmessage**

Since `MyVUSchedule1.testsuite` is binary, the remote and local changes must have to be merged manually. If necessary, you must first discuss with the user who authored or committed the changes. If their changes are extensive and the locally performed changes are relatively minor, you can overwrite the local version with the remote by selecting **Theirs:commit ID commit message**. You can then do the local changes again.

Alternatively, if the local changes are extensive when compared to the changes in the remote branch, then you can overwrite the remote version by selecting the **Ours:commit ID commit message** option. Later, you must manually redo the remote changes on the local copy.



7. Redo the changes that were broken because of the replace action.

8. Move the files from **Unstaged Changes** to **Staged Changes** by using the **Add selected files to the index** or icon.

9. Enter a message in the **Commit Message** pane, and then click **Commit and Push**.

   The **Push Result** dialog box is displayed with the status of the commit to the remote Git repository.

Configuration of test runs as a GitHub Actions Workflow

You can configure test runs of test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI in a GitHub Actions Workflow. You can choose to run test resources that are in the project in the GitHub Workflow when you want to run the test resources without opening the Rational® Functional TesterHCL OneTest™ UI.

Before you configure a test run for any test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI, you must complete certain tasks. See Prerequisites to run tests in a GitHub Actions Workflow on page 346.

After you complete the prerequisite tasks, you can proceed to configure a test run in a GitHub Workflow. See Configuring a test run in a GitHub Actions Workflow on page 347.

Prerequisites to run tests in a GitHub Actions Workflow

Before you configure a test run for any test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI, you must complete certain tasks to run the test resources in a GitHub Actions Workflow.

You must be familiar with creating projects and running test resources from Rational® Functional TesterHCL OneTest™ UI.

You must complete the following tasks in Rational® Functional TesterHCL OneTest™ UI:

- Created a test in the UI Test perspective. See Testing in the UI Test perspective on page 532.
- Opened and viewed the details of the test resources that you want to run in a GitHub Actions Workflow. You must make a note of the test resource details that are in Rational® Functional TesterHCL OneTest™ UI.

  You must note the following details of the test resource that are to be provided in the YAML file for the GitHub Action:
  - The complete path to the Eclipse workspace.
  - The name of the project containing the test.
  - The relative path from the project to the test including the file name of the test. A test can be WebUI test, Compound test, or Accelerated Functional Test (AFT) suite.

You must complete the following tasks outside of Rational® Functional TesterHCL OneTest™ UI:

- Create an empty Git repository on any GitHub Enterprise server, if you do not already use Git repositories to store the test resources that you create. For more information about creating Git repositories, refer to Creating a new repository.

  If you are already using Git repositories to store test resources created, you can use the same repositories.

  Note: You must have the required privileges in the Git repository to create a GitHub Actions Workflow.

- Enable the GitHub Actions option in the Git repository, if it is not available as an option to use. For more information about enabling the GitHub Actions, refer to Learn about GitHub Actions.
- Set up self-hosted runners on your computer. For more information, refer to Adding self-hosted runners.

You can continue to configure a run as a GitHub action, see Configuring a test run in a GitHub Actions Workflow on page 347.

Configuring a test run in a GitHub Actions Workflow

After you complete the prerequisite tasks to run tests in a GitHub Actions Workflow, you can configure the test run in a GitHub Actions Workflow.

Before you begin

You must have completed the tasks mentioned in Prerequisites to run tests in a GitHub Actions Workflow on page 346.

1. Perform the following steps to open the Git repository that you use or created, which has the GitHub Actions option enabled:

   a. Open the Git repository on your GitHub Enterprise server.

   b. Create a folder with the name `.github/workflows` in the root folder of the repository.

c. Create an empty YAML file in the `.github/workflows` folder.

The YAML file will be used to define the GitHub Action in the Actions Workflow.

2. Perform the following steps to define the YAML file for the GitHub Action:

a. Open the GitHub Marketplace (https://github.com/marketplace?type=) in a web browser.

b. Search for **Rational Functional Tester - UI Test GitHub ActionHCL OneTest WebUI GitHub Action** to run test resources that are in a Rational® Functional TesterHCL OneTest™ UI project.

c. Open **Rational Functional Tester - UI Test GitHub ActionHCL OneTest WebUI GitHub Action**.

d. Read through the information provided.

e. Copy the contents of the YAML file that is provided in the example by clicking the copy icon.

f. Open the YAML file that is in the `.github/workflows` folder of your repository and paste the copied contents.

g. Refer to the following table to enter the required details in the YAML file:

Field	Description
Required	
workspace	Enter the complete path of the Eclipse workspace.
project	Enter the name of the project containing the test.
suite	Enter the name of the test that you want to run.
Optional	
imshared	Enter the complete path to the location of the IBMIMSharedHCLIMShared folder if it is not in the default path. For example, `D:\IBM\IBMIMShared-D:\HCL\HCLIMShared` **Note:** If you do not enter any value, the default path of the IBMIMSharedHCLIMShared folder is considered for test run. The default path in the different operating systems is as follows:

Field	Description
	 ▪ Linux: `/opt/IBM/IBMIMShared/opt/HCL/HCLIMShared` ▪ Mac: `/Application/IBM/IBMIMShared/Application/HCL/HCLIMShared` ▪ Windows: `C:\Program Files\IBM\IBMIMSharedC:\Program Files\HCL\HCLIMShared`
configFile	Enter the complete path of the file that contains the parameters for a test run.
exportLog	Enter the file path to store the exported HTTP test log. You can provide multiple parameter entries when running multiple tests. You must use a colon to separate the parameter entries. For example: `c:/logexport.txt:c:/secondlogexport.txt`
exportReport	Use this option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. For example, to export the report to only the PDF format, you can input the following values: *type=unified;format=pdf;folder=Exportedreport102;filename=testreport*
exportStatReportList	Enter the list of report IDs separated by a comma. You must use this option with **exportstats** or **exportstatshtml** to list the reports that you want to export in place of the default reports, or the reports selected under **Preferences**. Perform the following steps to retrieve the report IDs: ▪ Navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports** in Rational® Functional TesterHCL OneTest™ UI. ▪ Select the required reports in the **Select reports to export** section. ▪ Click **Copy ID to clipboard**.
exportStats	Enter the complete path of the folder to store the exported statistical report data.
exportStatsFormat	Specify a format for the report that you want to export. You must use the **exportStats** parameter along with the **exportStatsFormat** parameter. You must use at least one of the following formats: ▪ *simple.csv* ▪ *full.csv*

Field	Description

	▪ *simple.json*
	▪ *full.json*
	▪ *csv*
	▪ *json*
	For example,
	`Export stats = <local_dir_path>`
	`Export stats format = json`
	You can add multiple formats for the report separated by a comma (,).
	The reports are saved to the location specified in the **exportStats** field.

exportStatsHtml	Enter the complete path to a directory to export web analytic results. If you run multiple tests, do not provide a value in this field.
imports	Enter the complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets.
labels	Enter the labels to be added to the test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, *"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset.
overwrite	Set the value as `true` to overwrite the existing result file and retain the same file name. The default value, false, indicates that the new result file is created.
protocolinput	You can use this option with additional arguments as follows:

Field	Description
	▪ To run a Web UI test in parallel on different browsers **-protocolinput** *"all.available.targets.in.parallel=all"* **Note:** If you use the **-protocolinput** argument, you must not use the following equivalent **-vmargs** arguments: `-vmargs "-Dall.available.targets.in.parallel=all"` ▪ To specify the Web UI preferences such as highlighting the page element and capturing screens For example, **-protocolinput** *"webui.highlight=<value>;webui.report-.screenshots=<value>"* where *webui.highlight* specifies whether the page element must be highlighted and *webui.report* specifies whether the screens must be captured while playing back the test in the browser. ▪ To run only the failed tests from a previous playback in an Accelerated Functional Test suite **cmdline -workspace** *workspacename* **-project** *projectname* **-aftsuite** *aftsuitname* **-exportlog** *exportlogpath* **-results** *autoresults* **-protocolinput** *"runfailedtests=true"* In the preceding example, *runfailedtests=true* specifies whether the failed test from a previous playback must be rerun in Accelerated Functional Test suite. ▪ To automatically resolve the browser and driver incompatibility, while you play back the Web UI tests **-protocolinput** *"webui.browser.driver.autoupdate=true"* ▪ To apply guided-healing and self-healing features while you run Web UI tests **cmdline -workspace** *workspacename* **-project** *projectname* **-suite** *test1* **-exportlog** *exportlogpath* **-results** *autoresults* **-protocolinput** *"autoheal=true"*
publish	Specify the server URL and project name to publish test results to Rational® Test Automation ServerHCL OneTest™ Server.

Field	Description
	You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **publish** parameter: ▪ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace-.name**=*name_of_the _teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace-.alias**=*name_of_the _teamspace_alias* You must consider the following point while providing the values: ▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`. For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*. Where, %20 is the hexadecimal value of the space character.
publish_for	Specify the option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **publishFor** parameter along with the **publish** parameter. The following are the available options that you can use for the **publishFor** parameter: ▪ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict. ▪ **PASS** - You can use this option to publish test results for the tests that have passed. ▪ **FAIL** - You can use this option to publish test results for the tests that have failed. ▪ **ERROR** - You can use this option to publish test results for the tests that included errors. ▪ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.

Field	Description
	You can add multiple parameters separated by a comma.
publishReports	Specify the option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **publishReports** are as follows: ▪ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455. ▪ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469. ▪ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491. You must use the **publishReports** parameter along with the **publish** parameter.
results	Enter a folder name that is relative to the project to store the test results. For example, `results folder/resultname`.
swapDatasets	For a test, the default value is the dataset specified in the test editor. You must use the **swapdatasets** option to replace dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example: `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`.
User Comments	Enter the text within double quotation mark to display it in the **User Comments** row of the report.
varFile	Enter the complete path to the XML file that contains the variable name and value pairs.
vmArgs	To specify the Java™ maximum heap size for the Java™ process that controls the command line playback, use the **-vmargs** option with the *-Xmx* argument. For example, when you use**-vmargs** *-Xmx4096m*, specify a maximum heap size of 4096m. This method is similar to specifying *-Xmx4096m* in the eclipse.ini file for the workbench when playing back the test from the user interface.

Field	Description
	To collect the response time data for the app itself and for the server and network and display them in different bar charts, use **-vmargs** *"-De2e.collect=true"*. For desktop-based web applications, the response time data is captured and displayed by default. To execute tests in parallel on all mobile devices, which are in passive mode, connected to the workbench and ready for playback, use **-vmargs** *"-Dall.available.targets.in.parallel=true"*. To execute tests in parallel on all supported desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=all"*. To execute tests in parallel on selected desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=browser1,browswer2,browser3"*. You must separate browser names with a comma. For example, *firefox, ff, chrome, ie, ie64, safari, "-Dall.available.targets.in.parallel=browser1,browser2,browser3"*.

h. Save the YAML file.

3. Perform the following steps to set up a Runner for the GitHub Action in GitHub:

   a. Click the Settings icon  .

   b. Click **Actions > Runners > New self-hosted runner**.

   c. Follow the onscreen instructions to set up a self-hosted runner on your computer.

      After the runner is installed on your computer, you must start the runner. The runner is listed in the **Runners** pane in the Git repository. The runner is in the *Idle* state when it is connected to Github and is ready to execute jobs.

      Note: You must ensure that the runner is not in the *Offline* state before you run a workflow.

4. Perform the following steps to start the GitHub Action in the Workflow:

   a. Click the **Actions** option in the Git repository.

      The GitHub Action that you configured in the YAML file for Rational® Functional TesterHCL OneTest™ UI is displayed as an **Action** under **Workflows**.

   b. Click **Rational Functional Tester - UI Test GitHub ActionHCL OneTest WebUI GitHub Action**, and then click **Run workflow**.

Results

You have configured and run a resource that is in your project on Rational® Functional TesterHCL OneTest™ UI in a GitHub Action Workflow.

What to do next

You can perform any of the following tasks on the Git repository page by clicking the job that was completed:

- View the status of the test run by expanding the **Execute test** section.
- View the links to the test results and reports listed in the **Execute test** section.

Testing with IBM® Engineering Workflow Management

You can manage test assets with Jazz™ source control management by integrating the test workbench with IBM® Engineering Workflow Management (formerly known as Rational® Team Concert™) Eclipse client.

Introduction

You can use IBM® Engineering Workflow Management Eclipse client to connect to compatible Jazz™ servers, including IBM® Engineering Workflow Management servers. You must have a compatible version of IBM® Engineering Workflow Management server setup to use Jazz™ source control management. For information about compatible versions, see System Requirements on page 28.

Use this feature to do the following tasks:

- Access IBM® Engineering Workflow Management Eclipse client.
- Manage the test assets by using Jazz™ source control management.

If you have installed the IBM® Engineering Workflow Management Eclipse client, see the IBM® Engineering Workflow Management help to learn more about using IBM® Engineering Workflow Management.

To access work items, you can switch to the **Work Items** perspective by clicking **Window > Open Perspective > Work Items**.

Installing IBM® Engineering Workflow Management client

You must have obtained the installation package that is available on the IBM Passport Advantage site. For information about installation, see IBM® Engineering Workflow Management help.

Notes:

- From V7.0 or later, Rational® Team Concert™ is renamed to IBM® Engineering Workflow Management.

Tips:

- You must install IBM® Engineering Workflow Management by using Installation Manager in the same package group as the test workbench.
- Both the IBM® Engineering Workflow Management or IBM® Engineering Workflow Management Eclipse client and the test workbench must use the same workspace.

Tracking defects with IBM® Engineering Workflow Management

You can submit defects to IBM® Engineering Workflow Management from Rational® Functional TesterHCL OneTest™ UI. By default, the test log editor uses Bugzilla as the defect tracking site. You must configure the product to use Rational® Team Concert™ for defect tracking.

1. Click **Window > Preferences > Test > Test Log Editor**.
   Result
   The **Test Log Editor** preferences window opens.
2. Specify the URL of IBM® Engineering Workflow Management server in the **Submit URL**, **Search URL**, and **Open URL** fields.
   Contact the administrator of the IBM® Engineering Workflow Management server for more information.
   Example
   The following are the example of URLs, where the name of the IBM® Engineering Workflow Management server is *ewm.example.com* and the name of the project is *projectname*:

   Table 2. Example of EWM Server URL

Field names	Field values
Submit URL	`https://ewm.example.com:9443/jazz/web/projects/projectname#action=com.ibm.team.workitem.newWorkItem`
Search URL	`https://ewm.example.com:9443/jazz/web/projects/projectname#action=jazz.viewPage&id=com.ibm.team.workitem`
Open URL	`https://ewm.example.com::9443/jazz/web/projects/projectname#action=com.ibm.team.workitem.viewWorkItem&id=`

   Note: You must change the URLs, if there is change in name of the IBM® Engineering Workflow Management server.

Related information

Test log overview on page 1493

Testing with Engineering Test Management

You can integrate IBM® Engineering Test Management (formerly known as IBM® Rational® Quality Manager) with Rational® Functional TesterHCL OneTest™ UI to initiate test runs from Engineering Test Management.

To run tests from Engineering Test Management, you must configure the default adapter that is installed when you install Rational® Functional TesterHCL OneTest™ UI.

You can run the adapter in the following modes:

- GUI
- CLI

Engineering Test Management reports

When you run a test script from Engineering Test Management, the default report that is displayed during a test run is attached to the results of Engineering Test Management. You can customize the reports based on your requirements. See Customizing reports on page 1479.

If you use Engineering Test Management 4.0 or later, you can view and analyze the test reports in Engineering Test Management. You can analyze the test reports while the test is in running state and after the test run is complete. You can click the **Analyze Results Interactively using Rational Functional TesterHCL OneTest UI** option from the **Execution Results** dialog box to view the test reports in Engineering Test Management.

The result completion state that is reported to Engineering Test Management reflects the overall verdict of the test log that is associated with the run. See Test log overview on page 1493. In many cases, a test might contain a failed verification point, but still is considered as passed. You can view the attached report in the execution result of Engineering Test Management, and then set the execution results status accordingly.

You can view the full run results from within Rational® Functional TesterHCL OneTest™ UI by opening Rational® Functional TesterHCL OneTest™ UI in the workspace that is configured to be used by the adapter.

If the adapter is running from the command line, you must stop the adapter before opening Rational® Functional TesterHCL OneTest™ UI. When Rational® Functional TesterHCL OneTest™ UI is opened, you can access the full test reporting and test log capabilities. The test results for the runs that are initiated from Engineering Test Management are under the Engineering Test Management **Results** page.

Known limitations

- You cannot run tests from Engineering Test Management with encrypted datasets. When using such datasets, a password prompt is not displayed in the adapter service or in the command-line interface. The use of encrypted datasets are not recommended in the GUI mode, because it requires user interaction with Rational® Functional TesterHCL OneTest™ UI to initiate test runs from Engineering Test Management.

- You can start only one adapter per product installation on a given computer. If you use multiple adapters on the same computer, it requires you to install each product as its own software package in its own directory. If you want to run multiple adapters on the same computer, you must ensure that adapters are using different workspaces.

For information about using Engineering Test Management, refer to the IBM Engineering Lifecycle Management documentation.

Refer to the following topics to learn more about integrating Engineering Test Management with Rational® Functional TesterHCL OneTest™ UI.

Configuring the Engineering Test Management adapter

You must configure the Engineering Test Management adapter to establish a successful connection between Rational® Functional TesterHCL OneTest™ UI and Engineering Test Management.

Before you begin

You must have the following information:

- The URL of the Engineering Test Management server.
- A user credential and valid license to access Engineering Test Management.
- The user account must be added to the project area that is being accessed by the adapter with write permissions to the project.
- You must have the following information:
    - The URL of the Engineering Test Management server.
    - A user credential and valid license to access Engineering Test Management.
    - The user account must be added to the project area that is being accessed by the adapter with write permissions to the project.
- You must have added `-Dhptcostconfirm` and *-DCMDLINE_PORT=999* to the `eclipse.ini` file before you run a test orcompound test or accelerated functional test that consume VU-Hours from Engineering Test Management.

  The `eclipse.ini` file must be available at the installation directory of Rational® Functional TesterHCL OneTest™ UI.

For more information about Engineering Test Management, refer to the IBM Engineering Lifecycle Management documentation.

1. Reconfigure the JRE path if you are using Kerberos for authentication:
    a. Modify the JRE path to use the Oracle Java path as follows in the `eclipse.ini` file in the `C:\Program Files\IBM\SDPC:\Program Files\HCL\SDP` directory:

    Change `-vm jdk/jre/bin/j9vm/jvm.dll` to

    `-vm C:\Program Files\Java\jre1.8.0_331\bin\server\jvm.dll`.

    b. Navigate to the `FunctionalTester> RQMAdapter` directory in the Rational® Functional TesterHCL OneTest™ UI installation directory.
    c. Replace *%RFT_ARG_JRE%* with the Oracle Java path on your system in the following line of the `configureadapter.bat` file:

    ```
    start "" %RFT_ARG_JRE% -classpath %RFT_INSTALL_DIR%\rational_ft.jar
    com.rational.test.ft.RQMAdapterConfigure %*.start "" %RFT_ARG_JRE% -classpath %RFT_INSTALL_DIR
    %\hcl_ft.jar com.rational.test.ft.RQMAdapterConfigure %*.
    ```

    For example,

    ```
    start "" "C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe" -classpath %RFT_INSTALL_DIR
    %\rational_ft.jar com.rational.test.ft.RQMAdapterConfigure %*start "" "C:\Program
    Files\Java\jre1.8.0_321\bin\javaw.exe" -classpath %RFT_INSTALL_DIR%\hcl_ft.jar
    com.rational.test.ft.RQMAdapterConfigure %*
    ```

2. Open Rational® Functional TesterHCL OneTest™ UI.
3. Click **Window > Preferences > Quality Manager Adapter**.
4. Enter the following information of the Engineering Test Management:

Fields	Actions
Server URL	Enter the URL of Engineering Test Management. For example, `https://<hostname>:<portnumber>/qm` **Note:** If you rename the Engineering Test Management server, you must perform the following tasks:

Fields	Actions
	a. Update the Engineering Test Management server name in the hosts file with a new name. b. Update the **Server URL** field with the new name. c. Configure the adapter to point to the new URL.
Adapter name	Enter a unique name to identify the Engineering Test Management adapter. The Engineering Test Management adapter uses the name of the computer as the default name of the adapter.
Project area	Enter the name of the project area in Engineering Test Management.

5. Select one of the following **Authentication type** from the drop-down list to connect to Engineering Test Management:

Authentication type	Actions
Username and Password	Perform the following steps: a. Enter the username associated with Engineering Test Management in the **User ID** field. b. Enter the password associated with the username of Engineering Test Management in the **Password** field.
KERBEROS	Click **Browse** to locate and select the `kerberos.ini` file in the **Configuration File** field. **Note:** The `kerberos.ini` file is automatically created when you set up Kerberos. For example, on Windows systems, you can locate the file in the `c:\windows\krb5.ini`. The file name and the location might change based on the operating systems.
SSLCERT	Perform the following steps:

Authentication type	Actions
	a. Enter the location of the SSL certificate keystore in the **Certificate Location** filed. b. Enter the keystore password in the **Password** field. **Note:** The expected format of the keystore is p12. The keystore must contain the client certificate that the adapter uses when you authenticate with Engineering Test Management.
SMARTCARD	Select a certificate from the drop-down list from the **Certificate Selection** field.

6. **Optional:** Select the **Enable Proxy** checkbox to connect through a proxy computer and perform the following steps to enter the **Proxy Details** of the computer:
   a. Enter the hostname of the proxy computer in the **Host** field.
   b. Enter the port number of the proxy computer in the **Port** field.
   c. Enter the username and password of the proxy computer in the **User** and **Password** fields.
7. Click **Apply and Close** to save and close the configuration.

Results

You have configured the details of Engineering Test Management on Rational® Functional TesterHCL OneTest™ UI.

What to do next

You must start the adapter from Rational® Performance TesterHCL OneTest™ Performance, command-line interface, or as a Windows service.

OR

You must start the adapter either from Rational® Functional TesterHCL OneTest™ UI or command-line interface to run Web UI tests.

Related information

Connecting and disconnecting the Engineering Test Management adapter from the GUI mode on page 362

Starting and stopping the Engineering Test Management adapter from the command line on page 364

Starting and stopping the Engineering Test Management adapter as a Windows service

Importing test assets into Engineering Test Management on page 365

Configuring the workspace directory of the adapter

You must configure the workspace directory of the adapter to start or stop the Engineering Test Management adapter from command-line interface.

About this task

If the **Use resources that are local to a test machine** option is set in Engineering Test Management, then the WORKSPACE_DIR must be set to the same workspace where your test assets are located.

1. Locate the `adapter.config` file in the *`product_install_dir`*`\RPT-RST_RQMAdapter\config` `\`*`product_install_dir`*` \HOT-PERF_RQMAdapter\config\` directory.
   Where *product_install_dir* is the directory where Rational® Functional TesterHCL OneTest™ UI is installed.

   For example, `C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest`.
2. Edit the WORKSPACE_DIR variable in the `adapter.config` file to point to the same test workspace that you want the adapter to use.

   For example, `WORKSPACE_DIR= C:\Users\`*`username`*`\IBM\rationalsdp` `\`*`my_adapter_workspace`*`WORKSPACE_DIR= C:\Users\`*`username`*`\HCL\HCLOneTest` `\`*`my_adapter_workspace`*.

Results

You have configured the workspace directory of the adapter.

What to do next

You can start or stop the Engineering Test Management adapter either from command-line interface or as a Windows service.

Connecting and disconnecting the Engineering Test Management adapter from the GUI mode

You can use the **Quality Manager Adapter** view to connect, disconnect, and view adapter activities from Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have configured the Engineering Test Management adapter. See Configuring the Engineering Test Management adapter on page 358.

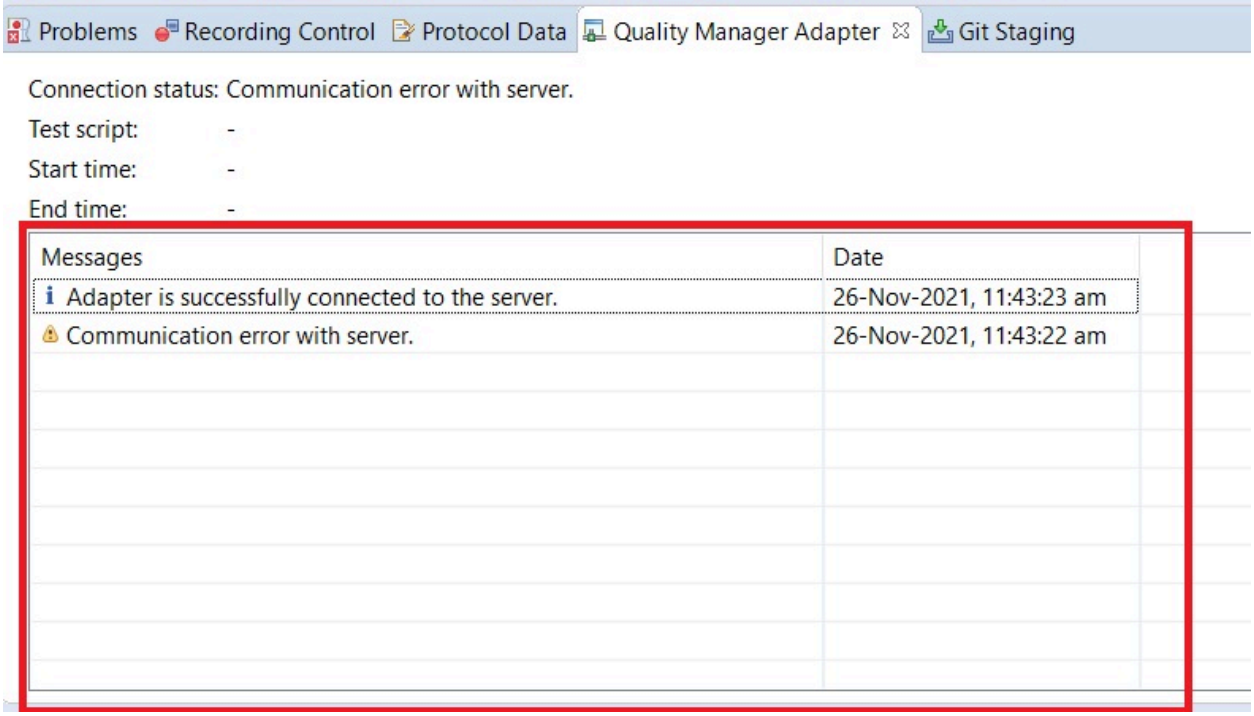
About this task

In the GUI mode, when a script is run from Engineering Test Management, you can see the test run in progress inside Rational® Functional TesterHCL OneTest™ UI as though the test were run manually in Rational® Functional TesterHCL OneTest™ UI.

Push buttons to connect and disconnect to the Engineering Test Management server are located in the upper-right corner of **Quality Manager Adapter** view. This view also has a local preferences menu that you can use to control some behavior of the GUI mode adapter. If you see errors or warnings, use the **Error Log** view for further investigation.

Note: You must not use Rational® Functional TesterHCL OneTest™ UI while the adapter is running. If you do so, you might interfere with the ability of the adapter to run test scripts. You must stop the adapter before you open Rational® Functional TesterHCL OneTest™ UI.

The following image displays the activities of the adapter in the **Quality Manager Adapter** view:



1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click **Window > Show View > Quality Manager Adapter**.
3. Perform the following actions either to connect or disconnect the adapter:
    - Click the **Connect to RQM** icon to connect the adapter.
    - Click the **Disconnect from RQM** icon to disconnect the adapter.

Results

You have connected or disconnected the Engineering Test Management adapter from Rational® Functional TesterHCL OneTest™ UI.

Related information

Configuring the Engineering Test Management adapter on page 358

Starting and stopping the Engineering Test Management adapter from the command line on page 364

Importing test assets into Engineering Test Management on page 365

Starting and stopping the Engineering Test Management adapter from the command line

You can use the command-line interface to start, stop, and view activities of the Engineering Test Management adapter that you configured in Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have performed the following tasks:

- Configured the adapter in Rational® Functional TesterHCL OneTest™ UI. See Configuring the Engineering Test Management adapter on page 358.
- Configured the workspace directory of the adapter. See Configuring the workspace directory of the adapter on page 362.

About this task

When you run test assets from the command-line interface, the adapter activities are printed to the `adapter.log` file that can be accessed from *`product_install_dir`*`\RFT_RQMAdapter\logs`*`product_install_dir`*`\HOT-UI_RQMAdapter\logs`.

To print the current status of the adapter, you must navigate to the *`product_install_dir`*`\RFT_RQMAdapter\bin`*`product_install_dir`*`\HOT-UI_RQMAdapter\bin` directory, and then you can run the RQMAdapter.bat STATUS command.

Where, *`product_install_dir`* is the installation directory of Rational® Functional TesterHCL OneTest™ UI.

Warning: You must not use Rational® Functional TesterHCL OneTest™ UI while the adapter is running. You must stop the adapter before you open Rational® Functional TesterHCL OneTest™ UI for any reason.

1. Open a command-line interface.
2. Navigate to the *`product_install_dir`*`\RPT-RST_RQMAdapter\bin\`*`product_install_dir`*`\HOT-PERF_RQMAdapter\bin\` directory.
3. Perform the following step either to start or stop the adapter:

- Run the following command to start the adapter from the command line:

Operating system	Command to be run
Windows™	RQMAdapter.bat START
Linux™	RQMAdapter.sh START

- Run the following command to stop the adapter from the command line:

Operating system	Command to be run
Windows™	RQMAdapter.bat STOP
Linux™	RQMAdapter.sh STOP

Results

You have started or stopped the Engineering Test Management adapter from the command-line interface.

Related information

Configuring the Engineering Test Management adapter on page 358

Connecting and disconnecting the Engineering Test Management adapter from the GUI mode on page 362

Importing test assets into Engineering Test Management on page 365

Importing test assets into Engineering Test Management

You can import the functional tests into Engineering Test Management by using an adapter.

Before you begin

The adapter must be running on a computer where the test assets are located.

About this task

From Rational® Functional TesterHCL OneTest™ UI, you cannot import AFT suites into Engineering Test Management because AFT suites are not supported with the Engineering Test Management integration.

The names of script types, Rational® Functional TesterHCL OneTest™ UI, and Rational® Test WorkbenchHCL OneTest™ Studio, are compatible with Rational® Functional TesterHCL OneTest™ UI, and Rational® Test WorkbenchHCL OneTest™ Studio products.

1. Log in to Engineering Test Management.
2. Click **Construction > Import Test Scripts**.
3. Select one of the following test scripts in the **Script Type** field:

a. Rational® Functional TesterHCL OneTest™ UI to import a functional test from Rational® Functional TesterHCL OneTest™ UI.
b. Rational® Test WorkbenchHCL OneTest™ Studio to import a Web UI test from Rational® Functional TesterHCL OneTest™ UI or import a test from Rational® Test WorkbenchHCL OneTest™ Studio.

4. Select **Use test resources that are local to a test machine**, and click **Select Adapter**.
5. Select the computer on which the adapter is running, and click **Next**.
6. Enter the name of the project in the **Project Path** field, and then click **Go**.



Note: You must specify only the project name and not the entire path to the project.

7. Select the test assets that you want to import, and then click **Finish**.
8. Select those test assets to import again, and then click **Import**.

Results

You have imported the test assets to Engineering Test Management by using the adapter.

Related information

Configuring the Engineering Test Management adapter on page 358

Connecting and disconnecting the Engineering Test Management adapter from the GUI mode on page 362

Starting and stopping the Engineering Test Management adapter from the command line on page 364

Testing shared assets with Rational Quality Manager on page 366

Testing shared assets with Rational® Quality Manager

You can make test projects and assets shareable in Rational® Quality Manager. By sharing assets, any computer with your product, that is connected to Rational® Quality Manager can execute a test or schedule.

Before you begin

When you are working with tests from a remote shared location, Rational® Functional TesterHCL OneTest™ UI uses a local workspace for the Rational® Quality Manager adapter. This adapter workspace is different from normal workspaces because the test assets are stored remotely. This means that every asset that is related to the test is downloaded from the shared location into the local workspace before execution. The following limitations apply:

- Assets in the adapter workspace might be deleted or overwritten with newer versions when updates are made to the shared location.
- If you change the shared location in the adapter workspace, the entire project is removed from the adapter workspace.
- Test results are stored in a different project, called `RQM_Results`, and are never deleted. The Rational® Quality Manager test result page links to the correct location.

Note: Do not edit test assets in the adapter workspace because you might lose your work. You must use these assets only for running tests.

If you are using source control and want to include only the minimum required assets, then include the following files:

- All `*.testsuite` tests files
- The `/src` directory if you use custom code
- All `*.dp` dataset files
- All `*.location` location files
- All digital certificates
- All WSDL and SOA security files

Note: All other assets, such as test results, are not required.

Custom code Java™ classes in the shared assets cannot use libraries that are outside the workspace. If your custom code must use such a library, then copy the library into the project, and update the classpath to use the local copy.

1. Create a shared directory on the computer that hosts the UNC file system that contains the test projects to share.
   Example
   For example, create a directory called: `C:\MyRemoteWorkspace`.
2. Copy the test projects to share into the shared directory.
   If a project is stored in source control software, then copy it from there.
3. Check that the Rational® Quality Manager server can access the shared directory by using UNC paths.
   Example
   For example, the `\\MyServer\RPTRemoteAssets\` path must be mapped to the `C:\MyRemoteWorkspace` directory.
4. In Rational® Quality Manager, specify the directory that contains the actual test projects that are located in the shared directory.

5. Verify that you have correctly specified the UNC shared directory by browsing for the shared resource. Ensure that the first dialog box contains the projects at the first level.
   You must not have intermediate directories between the UNC shared directory and the project directory.

Related information

Configuring the Engineering Test Management adapter on page 358

Importing test assets into Engineering Test Management on page 365

Troubleshooting: Unable to open unified reports

When you try to open the unified report that is published to the Engineering Test Management after the test run, the report fails to open in the browser.

Exemple

Solution:

Perform the following steps to open the unified report in a browser:

1. Log in to Engineering Test Management.
2. Click the **Administration** icon  in the upper-right corner of the header, and then select **Manage Project Properties**.
3. Click **Test Result Preference** in the **Properties** tab.
4. Select the **Open attachments in test results directly, without prompting users to open or save them** checkbox.

   Note: This is a one-time setup to enable the option to open reports.

5. Click the unified report link to open the report in a browser.

Testing with Jenkins

You can use the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin to run tests on a Jenkins server.

To automate testing with Jenkins, you must configure Jenkins primary server and Jenkins secondary server. This configuration provides a single Jenkins installation on the Jenkins primary server to host multiple Jenkins secondary server for building and running tests. For more information about the Jenkins primary and secondary server relationship, refer to the Jenkins documentation.

You must install the required version of the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server, and install Rational® Functional TesterHCL OneTest™ UI on the Jenkins secondary server, where you create tests.

You can use either the Jenkins **Freestyle** project or the **Pipeline** project to run test assets from Jenkins. With **Freestyle** project, you can create a build step from the Jenkins UI to run the test assets. Whereas the **Pipeline** project uses a simple text scripts based on the Groovy programming language. You can define pipeline scripts during the configuration of the **Pipeline** project by using any of the following methods:

- **Pipeline script**: In this method, you can directly enter a script in the field provided within the Jenkins UI or you can select a sample **Pipeline script** from the drop-down list.
- **Pipeline script from SCM**: In this method, you can create a script with a Groovy editor and then commit the script file into the Git repository.

For more information about the Jenkins Pipeline, refer to the Jenkins documentation.

Refer to the following topics to learn more about integrating Jenkins with Rational® Functional TesterHCL OneTest™ UI in the UI Test perspective:

Environment variables

You can add environment variables on the Jenkins server to run the Jenkins build by referring to environment variables.

You can add an environment variable on the Jenkins server by navigating to **Manage Jenkins > Configure System > Global properties**. You can enter the variable name by using any of the following methods for the corresponding text fields in the **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test** step:

- Use the dollar sign ($) followed by the variable name.

  For example, `$workspace`

- Use the dollar sign ($) followed by the variable name between braces.

  For example, `${workspace}`

The IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin uses the actual value while running the job.

For example, if you add the environment variable named `workspace` with the value `C:\Users\IBM\workspace1C:\Users\HCL\workspace1`, then you can use `$workspace` or `${workspace}` as input to the **Workspace** field when running tests. During the run time, `$workspace` or `${workspace}` is substituted with its corresponding value `C:\Users\IBM\workspace1C:\Users\HCL\workspace1`.

Build parameters for Jenkins

You can define the build parameters when you configure either **Freestyle** or **Pipeline** projects. You must map the build parameters with the parameters of **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test**. When you use the build parameters, Jenkins prompts you to enter a value for the defined parameter when you run the Jenkins job. Each parameter has a name and value.

Jenkins supports various types of build parameters. You can choose the type of parameter based on your requirements. Some of the commonly used types of parameters are as follows:

- Boolean
- Choice
- Credentials
- File
- String
- Multi-line String
- Password

After you add the parameter, the name that you provided for the parameter must be entered in the corresponding field in the **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test** step in the following format:

`${name_of_the_parameter}`

For example, `${workspace}`

Each time when you run a test from Jenkins, you might want to use a different configuration file. In such a case, you can add a **String Parameter** by selecting the **This project is parametrized** option from the **General** tab when you configure the project.

You can then provide a name for the string parameter as *Config_file_name* in the **Name** field. You can also provide a default value to be used in the **Default Value** field and describe the parameter in the **Description** field. These fields are optional fields.

When you provide the details of the test run in the **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test** step, you can enter the value for the **Config File** field as `${Config_file_name}`.

When you run the Jenkins job, the *Config_file_name* parameter prompts for you to specify the file name. You can provide the name of the configuration file to be used for the test run. You must then click **Build** to run the test.

Related information

Configuring the Freestyle project on page 372

Creating a pipeline script from Jenkins on page 381

Running tests from Jenkins on page 392

Task flows for running test assets from Jenkins

You can perform certain tasks to run test assets from the Jenkins **Freestyle** project or the **Pipeline** project.

The following table lists the task flows for running test assets from the Jenkins **Freestyle** project:

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 372
Configure the **Freestyle** project.	Configuring the Freestyle project on page 372
Run Rational® Functional TesterHCL OneTest™ UI UI tests on Jenkins.	Running tests from Jenkins on page 392

You can perform the following tasks to run test assets from the Jenkins **Pipeline** project by using the sample script:

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 372
Create a pipeline script from the Jenkins UI.	Creating a pipeline script from Jenkins on page 381
Configure the **Pipeline** project.	Configuring the Pipeline project by using the sample script on page 389
Run Rational® Functional TesterHCL OneTest™ UI UI tests on Jenkins.	Running tests from Jenkins on page 392

You can perform the following tasks to run test assets from the Jenkins **Pipeline** project by using the script from Source Code Management (SCM):

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 372
Create a pipeline script and commit the script to the Git repository.	For more information, refer to the Jenkins documentation.
Configure the **Pipeline** project.	Configuring the Pipeline project by using the script from SCM on page 391
Run Rational® Functional TesterHCL OneTest™ UI UI tests on Jenkins.	Running tests from Jenkins on page 392

Installing the plugin on the Jenkins primary server

You must install the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin to run UI test assets from the Jenkins server.

Before you begin

You must have completed the following tasks:

- Verified that you have a Jenkins primary server and secondary server.
- Downloaded the Rational® Functional TesterHCL OneTest™ UI Web UI Jenkins plugin `RFT-WebUI-Jenkins-11.09.0` from the IBM WebSphere, Liberty & DevOps Community portal HCL® License & Delivery portal.

1. Log in to the Jenkins server.
   Result
   The Jenkins dashboard is displayed.
2. Click **Manage Jenkins > Manage plugins**, and then click **Advanced** tab.
3. Click **Choose File** and then locate and open the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin.
4. Click **Upload**.
   Result
   The IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin is displayed in the **Installed** tab.
5. Perform the following steps to provide Random TCP Ports for Java™ Network Launch Protocol (JNLP) agents:
   a. Click **Manage Jenkins** from the Jenkins dashboard.
   b. Click **Configure Global Security** from the **Security** section.
   c. Click **Random** from the **Agents** section.
   d. Click **Save** to save and apply the changes.

Results

You have installed the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server.

What to do next

You can run the test from the Jenkins server. See Running tests from Jenkins on page 392.

Configuring the Freestyle project

You must configure a **Freestyle** project to add a build step, and then run test assets from Jenkins.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 372.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.
- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.
- Created a Jenkins **Freestyle** project.

About this task

When you create a **Freestyle** project in the Jenkins server, you must select the **Restrict where this project can be run** checkbox and enter the name of the labels that you provided during the creation of Agent in the **Label Expression** field.

1. Open the Jenkins **Freestyle** project, and then click **Configure**.
2. Click the **Build** tab, and then click **Add build step**.
3. Select the **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test** option from the drop-down list.
4. Provide the details about the test run for the fields as listed in the following table.

   You must provide the values for the following required fields:
   - **Name**
   - **Workspace**
   - **Project**
   - **Test Suite Name**

   If you include the required parameters such as **Workspace**, **Project**, and **Test Suite Name** in a configuration file and use the **Config File** field to specify the complete file path, then the values for these three fields are not required.

Field	Description
Required	
Name	The name of the Jenkins build step.
Workspace	The complete path to the Eclipse workspace.
Project	The path, including the file name of the project relative to the workspace.
Test Suite Name	The path, including the file name of the test to run related to the project.

Field	Description
	Note: You must provide the file name along with the file extension if you are using an Accelerated Functional Test suite. To run multiple tests of the same project sequentially, you must specify the test names separated by a `comma`.
Config File	The complete path to a file that contains the parameters for a test run. For example, `E:\Workspace1\Project1\Tests\config_file1.txt` For more information about creating command line config file, see Creating a command-line config file on page 1400. **Remember:** If you include the required parameters such as **Workspace**, **Project**, and **Test Suite Name** in a configuration file and use the **Config File** field to specify the complete file path, then the values for these three fields are not required.
Optional	
IMShared Location	The complete path to IBMIMSharedHCLIMShared location, if it is not the default location.
Import	The complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets.
Var File	The complete path to the XML file that contains the variable name and value pairs.
Results File	The name of the results file. The default result file is the test or schedule name with a time-stamp appended. The results file is stored in the **Results** directory. If you are running multiple tests, do not provide a name for the results file.
Overwrite Results File	Determines whether a result file with the same name is overwritten. The default value is `true`, which means the result file can be overwritten.
VM Args	The option to pass Java™ virtual machine arguments.
Protocol Input	The option to run a Web UI test in parallel on different browsers.

Field	Description

	For example, **Protocol Input** is *all.available.targets.in.parallel=all***Protocol Input** is *all.available.targets.in.parallel=chrome,ff,ie*
	
	Note: If you use the **Protocol Input** argument, you must not use the equivalent **VM Args** arguments:
	`VM Args = -Dall.available.targets.in.parallel=all`
	`VM Args = -Dall.available.targets.in.parallel=browser1,browser2,browser3`

Dataset Override	For a test, the default value is the dataset specified in the test editor. **Note:** You must use the **Dataset Override** option to replace the dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example, `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv` You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a `semicolon (;)`.
Labels	The option to adds labels to test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, *"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:**

Field	Description
	◦ When you run tests by using the double quotation marks ("") for the **Labels** field, then the labels in the test result do not include the double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field. ◦ When you use the **Config File** field to run tests, then labels provided in the configuration file take precedence over the labels provided in the **Labels** field.
Exported Statistical Report Data File	The complete path to a directory to store exported statistical report data. If you do not supply a value for `Exported Statistical Report Data File`, the logs will be saved in Jenkins `workspace/temp` directory.
Custom Report Format Files	A comma-separated list of absolute paths to custom report format files (.view files) to use when exporting statistical report data with the Export Statistical Report Data File option.
Exported Statistical Report in html	The complete path to a directory to export web analytic results. Analyze the results on a web browser without using the test workbench. If you run multiple tests, do not provide a value in this field. The web analytic results will be exported to the Jenkins workspace.
User Comments	The text to be displayed in the **User Comments** row of the report. Add text within the double quotation mark (*"*") **Note:** ◦ When you run tests by using the double quotation marks ("") for the **User Comments** field, then the **User Comments** row of a report does not contain double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field.
Export stats format	The field to specify a format for the report that you want to export. You must use the **Exported Statistical Report Data File** parameter along with the **Export stats format** parameter. You must use at least one of the following formats: ◦ *simple.csv* ◦ *full.csv*

Field	Description

	◦ *simple.json*
	◦ *full.json*
	◦ *csv*
	◦ *json*
	For example,
	`Exported Statistical Report Data File = <local_dir_path>`
	`Export stats format = json`
	You can add multiple formats for the report separated by a comma (,).
	If you want to export both the simple and full reports in a *json* or *csv* format, you can specify *json* or *csv* as the format in the field.
	The reports are saved to the location specified in the **Exported Statistical Report Data File** field.

Export Report	The option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. After you select this checkbox, you must choose the following details from the drop-down lists and enter the details: ◦ **Type**: Select **unified** from the list. ◦ **Format**: Specify the supported file formats such as xml, pdf, and html to export the unified report. You can also specify multiple formats separated by a comma to export the unified report simultaneously to different file formats. For example, to export the report to all the supported formats, you can use the following values: pdf,xml,html ◦ **Directory**: Enter the directory path where you want to save the exported file. ◦ **File Name**: Enter a name for the exported file.

Field	Description
	Remember: You must provide the values in the required fields such as **Workspace**, **Project**, and **Test Suite Name** to export the unified report. You must ensure not to use the **Config File** parameter with the **Export Report** parameter.

Publish	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server.
	
	You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **Publish** parameter in the test script.
	
	Use the following arguments with the **Publish** parameter:
	
	◦ To specify the project name, use any of the following formats:
	
	▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _-teamspace*
	
	▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _-teamspace_alias*
	
	You must consider the following points while providing the project name:
	
	▪ If the project name is not specified, then the value of the **Project** parameter is used.
	
	▪ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias==***name_of_the_teamspace_alias* options.
	
	For example:
	
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish =`
	` https://localhost:5443#project.name=test&teamspace.name=ts1`
	
	Where:
	
	▪ https://localhost:5443 is the URL of the server.
	
	▪ test is the name of the project.
	
	▪ ts1 is the name of the team space.
	
	▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`.

Field	Description

	For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*.
	
	Where, %20 is the hexadecimal value of the space character.
	
	◦ To avoid publishing of reports, use **no**.
	
	You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
	
	For example:
	
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = no`
	
	If you do not use the **Config File** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
	
	The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:
	
	◦ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**.
	
	◦ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).

PublishFor	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **PublishFor** parameter along with the **Publish** parameter. The following are the available options that you can use for the **PublishFor** parameter: ◦ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict. ◦ **PASS** - You can use this option to publish test results for the tests that have passed. ◦ **FAIL** - You can use this option to publish test results for the tests that have failed.

Field	Description

	◦ **ERROR** - You can use this option to publish test results for the tests that included errors.
	◦ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.
	You can add multiple parameters separated by a comma.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`PublishFor = FAIL,ERROR`

Publish reports	The option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **Publish reports** are as follows:
	◦ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455.
	◦ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.
	◦ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
	
	You must use the **Publish reports** parameter along with the **Publish** parameter.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish reports = STATS, TESTLOG`
	
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	
	For example,
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish reports = !TESTLOG`

Field	Description
	All the reports except the TESTLOG report is published to Rational® Test Automation Server-HCL OneTest™ Server after executing the command.

5. **Optional:** Click **Add build step** again, and provide details for the next test to run multiple tests under the same job.
6. Click **Save**.

Results

You have configured the **Freestyle** project by adding the build step.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 392.

Creating a pipeline script from Jenkins

You must create a pipeline script to run test assets from the Jenkins **Pipeline** project.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 372.
- Created a Jenkins **Pipeline** project.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Click **Pipeline Syntax** to generate a pipeline script.
4. Select the **step: General Build Step** option from the drop-down list in the **Sample Step** field.
5. Select the **Run IBM Rational Functional Tester - UI TestRun HCL OneTest UI - UI Test** option from the drop-down list in the **Build Step** field.
6. Provide the details about the test run for the fields as listed in the following table.

   You must provide the values for the following required fields:
   - **Name**
   - **Workspace**
   - **Project**
   - **Test Suite Name**

   If you include the required parameters such as **Workspace**, **Project**, and **Test Suite Name** in a configuration file and use the **Config File** field to specify the complete file path, then the values for these three fields are not required.

Field	Description
Required	
Name	The name of the Jenkins build step.
Workspace	The complete path to the Eclipse workspace.
Project	The path, including the file name of the project relative to the workspace.
Test Suite Name	The path, including the file name of the test to run related to the project. **Note:** You must provide the file name along with the file extension if you are using an Accelerated Functional Test suite. To run multiple tests of the same project sequentially, you must specify the test names separated by a `comma`.
Config File	The complete path to a file that contains the parameters for a test run. For example, `E:\Workspace1\Project1\Tests\config_file1.txt` For more information about creating command line config file, see Creating a command-line config file on page 1400. **Remember:** If you include the required parameters such as **Workspace**, **Project**, and **Test Suite Name** in a configuration file and use the **Config File** field to specify the complete file path, then the values for these three fields are not required.
Optional	
IMShared Location	The complete path to IBMIMSharedHCLIMShared location, if it is not the default location.
Import	The complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets.
Var File	The complete path to the XML file that contains the variable name and value pairs.
Results File	The name of the results file.

Field	Description
	The default result file is the test or schedule name with a time-stamp appended. The results file is stored in the **Results** directory. If you are running multiple tests, do not provide a name for the results file.
Overwrite Results File	Determines whether a result file with the same name is overwritten. The default value is `true`, which means the result file can be overwritten.
VM Args	The option to pass Java™ virtual machine arguments.

Protocol Input	The option to run a Web UI test in parallel on different browsers.
	
	For example, **Protocol Input** is *all.available.targets.in.parallel=all***Protocol Input** is *all.available.targets.in.parallel=chrome,ff,ie*
	
	Note: If you use the **Protocol Input** argument, you must not use the equivalent **VM Args** arguments:
	`VM Args = -Dall.available.targets.in.parallel=all`
	`VM Args = -Dall.available.targets.in.parallel=browser1,browser2,browser3`

Dataset Override	For a test, the default value is the dataset specified in the test editor. **Note:** You must use the **Dataset Override** option to replace the dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example, `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv` You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a `semicolon (;)`.
Labels	The option to adds labels to test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, *"label1, label2"*

Field	Description
	When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:** ◦ When you run tests by using the double quotation marks ("") for the **Labels** field, then the labels in the test result do not include the double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field. ◦ When you use the **Config File** field to run tests, then labels provided in the configuration file take precedence over the labels provided in the **Labels** field.
Exported Statistical Report Data File	The complete path to a directory to store exported statistical report data. If you do not supply a value for `Exported Statistical Report Data File`, the logs will be saved in Jenkins `workspace/temp` directory.
Custom Report Format Files	A comma-separated list of absolute paths to custom report format files (.view files) to use when exporting statistical report data with the Export Statistical Report Data File option.
Exported Statistical Report in html	The complete path to a directory to export web analytic results. Analyze the results on a web browser without using the test workbench. If you run multiple tests, do not provide a value in this field. The web analytic results will be exported to the Jenkins workspace.
User Comments	The text to be displayed in the **User Comments** row of the report. Add text within the double quotation mark (“”) **Note:**

Field	Description
	◦ When you run tests by using the double quotation marks ("") for the **User Comments** field, then the **User Comments** row of a report does not contain double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field.

Export stats format	The field to specify a format for the report that you want to export. You must use the **Exported Statistical Report Data File** parameter along with the **Export stats format** parameter.
	You must use at least one of the following formats:
	◦ *simple.csv*
	◦ *full.csv*
	◦ *simple.json*
	◦ *full.json*
	◦ *csv*
	◦ *json*
	For example,
	`Exported Statistical Report Data File = <local_dir_path>`
	`Export stats format = json`
	You can add multiple formats for the report separated by a comma (,).
	If you want to export both the simple and full reports in a *json* or *csv* format, you can specify *json* or *csv* as the format in the field.
	The reports are saved to the location specified in the **Exported Statistical Report Data File** field.

Export Report	The option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. After you select this checkbox, you must choose the following details from the drop-down lists and enter the details:

Field	Description
	◦ **Type**: Select **unified** from the list. ◦ **Format**: Specify the supported file formats such as xml, pdf, and html to export the unified report. You can also specify multiple formats separated by a comma to export the unified report simultaneously to different file formats. For example, to export the report to all the supported formats, you can use the following values: pdf,xml,html ◦ **Directory**: Enter the directory path where you want to save the exported file. ◦ **File Name**: Enter a name for the exported file. **Remember:** You must provide the values in the required fields such as **Workspace**, **Project**, and **Test Suite Name** to export the unified report. You must ensure not to use the **Config File** parameter with the **Export Report** parameter.
Publish	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **Publish** parameter in the test script. Use the following arguments with the **Publish** parameter: ◦ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias* You must consider the following points while providing the project name: ▪ If the project name is not specified, then the value of the **Project** parameter is used. ▪ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias==***name_of_the_teamspace_alias* options. For example:

Field	Description
	``` Workspace = C:/Users/IBM/rationalsdp/workspace1 Project = proj1 Test Suite Name = Tests/testHttp.testsuite Publish = https://localhost:5443#project.name=test&teamspace.name=ts1 ``` Where: ▪ https://localhost:5443 is the URL of the server. ▪ test is the name of the project. ▪ ts1 is the name of the team space. ▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`. For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*. Where, %20 is the hexadecimal value of the space character. ◦ To avoid publishing of reports, use **no**. You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them. For example: ``` Workspace = C:/Users/IBM/rationalsdp/workspace1 Project = proj1 Test Suite Name = Tests/testHttp.testsuite Publish = no ``` If you do not use the **Config File** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**). The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:

Field	Description
	◦ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**. ◦ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).

PublishFor	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **PublishFor** parameter along with the **Publish** parameter.
	The following are the available options that you can use for the **PublishFor** parameter:
	◦ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict.
	◦ **PASS** - You can use this option to publish test results for the tests that have passed.
	◦ **FAIL** - You can use this option to publish test results for the tests that have failed.
	◦ **ERROR** - You can use this option to publish test results for the tests that included errors.
	◦ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.
	You can add multiple parameters separated by a comma.
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`PublishFor = FAIL,ERROR`

Publish reports	The option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **Publish reports** are as follows: ◦ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455. ◦ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469. ◦ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491. You must use the **Publish reports** parameter along with the **Publish** parameter.

Field	Description

	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish reports = STATS, TESTLOG`
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	For example,
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish reports = !TESTLOG`
	All the reports except the TESTLOG report is published to Rational® Test Automation Server-HCL OneTest™ Server after executing the command.

7. Click **Generate Pipeline Script** and copy the script that you created for the build step to use it in the sample script.

Note:

You need this generated script to enter in step 5.c on page 390 in the following topic:

Configuring the Pipeline project by using the sample script

Results

You have created the pipeline script.

What to do next

You can configure the **Pipeline** project by using the **Pipeline script** option. See Configuring the Pipeline project by using the sample script on page 389.

Configuring the Pipeline project by using the sample script

You can configure a **Pipeline** project directly from the Jenkins UI by using the sample pipeline scripts to run test assets from Jenkins.

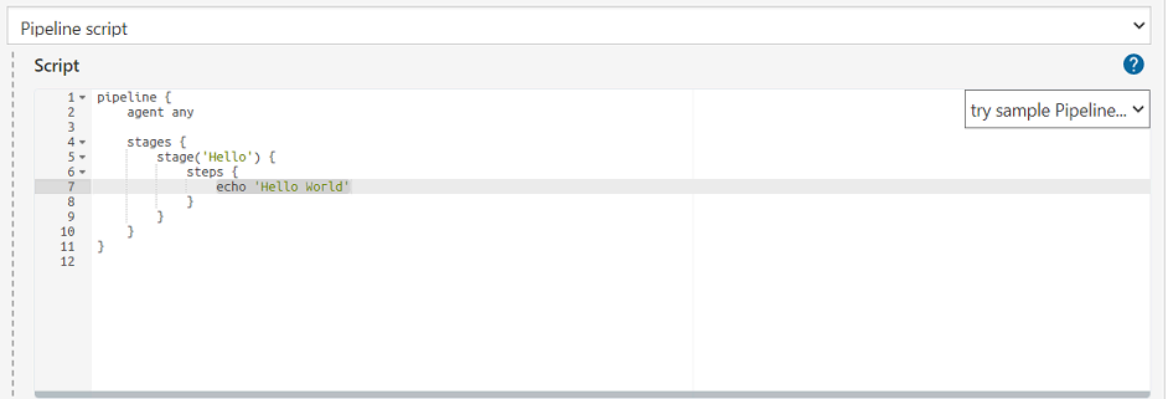
Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 372.
- Created a pipeline script. See Creating a pipeline script from Jenkins on page 381.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.
- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Select the **Pipeline script** option in the **Definition** field.
4. Select the sample script from the drop-down list.

   For example, if you select `Hello Word` from the drop-down list, then the sample script is displayed as follows:



5. Perform the following steps to edit the sample script:

   a. Replace the agent `any` in the script with the name of the label that you provided when you created the Agent.
   For example, if you provided a name of the label as `win10_1022`, then edit the script as follows:

   ```
   agent {label 'win10_1022'}
   ```

   b. Provide a name for the `stage` in the script.
   For example, `stage('Run FTUI Demo')`

   c. Specify the script that you created in step 7 on page 389 in place of *"echo 'Hello Word'"*.

6. Click **Save**.

Results

You have configured the **Pipeline** project by using the **Pipeline script** option.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 392.

Configuring the Pipeline project by using the script from SCM

You can configure the **Pipeline** project to use the pipeline script from the Source Code Management (SCM), and then run test assets from the Jenkins **Pipeline** project.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - UI Test Jenkins plugin on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 372.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.
- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.
- Created a pipeline script and committed the script to the Git repository. For more information, refer to the Jenkins documentation.
- Been granted administrator permission of the Jenkins server to add credentials into Jenkins.
- Added your GitHub credentials into Jenkins. For more information about adding global credentials, refer to the Jenkins documentation.
- Created a Jenkins **Pipeline** project.

About this task

If your pipeline scripts are complex, you can then write the script outside of Jenkins UI. You must commit that script file into your Git repository. During the **Pipeline** build process, Jenkins checks out the script file from the Git repository and runs your **Pipeline** project.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Select the **Pipeline script from SCM** option in the **Definition** field.
4. Select the **Git** option from the drop-down list in the **SCM** field.
5. Enter the URL of the Git repository in the **Repository URL** field where you stored your pipeline script.
6. Select the credentials of the Git repository from the drop-down list.
7. **Optional:** Enter the branch name of the Git repository in the **Branch Specifier** field.
8. Enter the path of the script that you stored in the Git repository in the **Script path** field.
9. Click **Save**.

Results

You have configured the **Pipeline** project by using the **Pipeline script from the SCM** option.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 392.

Running tests from Jenkins

You can run test assets either from the Jenkins **Freestyle** or the **Pipeline** project on the Jenkins server to test an application under test.

Before you begin

You must have completed the following tasks:

- Verified that you have test assets residing within Rational® Functional TesterHCL OneTest™ UI.
- Configured the **Freestyle** project, if you want to run test assets from the **Freestyle** project. See Configuring the Freestyle project on page 372.
- Configured the **Pipeline** project either by using the **Pipeline script** or **Pipeline script from SCM**, if you want to run test assets from the **Pipeline** project. See Configuring the Pipeline project by using the sample script on page 389 or Configuring the Pipeline project by using the script from SCM on page 391.
- Downloaded the latest **Test Results Analyzer** plugin from the Jenkinsci repository. For more information about the **Test Results Analyzer** plugin, refer to the Jenkins documentation.
- Installed the **Test Results Analyzer** plugin on the Jenkins server. For more information about installing the plugin, refer to the Jenkins documentation.
- Closed Rational® Functional TesterHCL OneTest™ UI.

About this task

After the test run is complete, you can view the test results in a tabular format when you use the Jenkins **Test Results Analyzer** plugin. With a TRA-based report, you can filter the results based on the status of the build. The status of the builds can be PASSED, FAILED, and SKIPPED. You can also identify the failed build from the TRA-based report and take the required action to resolve the build errors. The TRA report also represents the build information in **Line**, **Bar**, and **Pie** graphs so that you can easily read the data.

Note: You can view the TRA-based reports only when you run Web UI tests.

1. Log in to the Jenkins server.
   Result
   The Jenkins dashboard is displayed.
2. Open your Jenkins **Freestyle** or **Pipeline** project from the list.
3. Perform one of the following steps to run the test assets from Jenkins:
   - Click **Build Now**.
   - Click **Build with Parameters** to provide the values for the defined parameters, and then click **Build**.

Results

You have run the test from the Jenkins server.

What to do next

You can view the build logs by clicking the build number from the **Build History** pane, and then selecting the **Console Output** option.

You can also view the history of test results in a tabular format by clicking the **Test Results Analyzer** option. The TRA reports are available only for Web UI test assets.

Testing with Maven

Starting from 9.2.0, you can use the Maven plug-in that is provided with the testing product to run tests as part of your Maven build. Apache Maven is a software build tool based on the concept of a project object model (POM).

Before you begin

- You must have installed Rational® Functional TesterHCL OneTest™ UI and set an environment variable that points to the installation location.

  For Mac OS, add an environment variable that points to the installation directory of the product: `export TEST_WORKBENCH_HOME=/opt/IBM/SDPexport TEST_WORKBENCH_HOME=/opt/HCL/HCLOneTest`

  For Windows™ and Linux®, this environment variable is set when you install the product.
- You must have installed Maven from V3.2.0 and set up an environment variable that points to the `M2_HOME` installation directory.

Introduction

To automate testing with Maven, you must configure a pom.xml file and launch your tests from the command line using Maven command. You can either use your own pom.file, or one that is delivered with the product.

Three files are delivered with the product installation in the `<product install location>\SDP \maven2\<product install location>\HCLOneTest\maven2\` folder:

- `pomCustomSurefireSample.xml` for Windows, Linux and macOS.
- `pomMojoExecPluginSample_Linux.xml` for Linux and MacOS.
- `pomMojoExecPluginSample_Windows.xml` for Windows.

The files contain all types of dependencies as well as arguments required to execute the test scripts. For information on the supported arguments, see Supported options in Maven on page 395.

There are two methods to run tests with Maven.

Method 1

With this method, you can run one or several tests. If you use your own pom.xml file, edit it with the following lines and indicate which test(s) must be executed, otherwise, use the `pomCustomSurefireSample.xml` file as follows:

- Copy the `pomCustomSurefireSample.xml` to a directory.
- Edit the file and update the lines, enter the name and location of the test(s) that must be run. If the product is installed on a different drive or a different location, or if `IBMIMSharedHCLIMShared` location has been changed, enter the correct path to the `IBMIMSharedHCLIMShared` plug-in folder. For aftsuite attribute, you can input aft xml file as the parameter value.

```
<!--test suite="soa" project="AA" plugins="C:/Program
 Files/IBM/IBMIMShared/plugins"/-->"C:/Program Files/HCL/HCLIMShared/plugins"/-->
 importzip="c:/MavenTrial/Tests.zip" workspace="C:/TmpDir"/--> <!-- To publish, the user
 need to place the offline token authentication in the environment before --> <!--test
 suite="Wikipedia.testsuite" project="OutOfWsp" workspace="C:/Runtimes/runtime-RptMvn"
 publish="https://<Server>#project.name=<ProjectToPublish>" publish_for="ALL"/--><!--
 When the project name exists in two different team spaces, the conflict can be
 solved be adding teamspace.alias or teamspace.name as shown below --><!--test
 suite="Wikipedia.testsuite" project="OutOfWsp" workspace="C:/Runtimes/runtime-RptMvn"
 publish="https://<Server>#project.name=<ProjectToPublish>&amp;teamspace.alias=<TeamSpaceAlias>"
 publish_for="ALL"/--><!--test suite="Wikipedia.testsuite"
 project="OutOfWsp" workspace="C:/Runtimes/runtime-RptMvn"
 publish="https://<Server>#project.name=<ProjectToPublish>&amp;teamspace.name=<TeamSpaceName>"
 publish_for="ALL"/--><!--test suite="testSources/Test1.testsuite"/-->
```

- Run Maven to update the pom file version command and use the plug-in version currently available on delivered repositories.

```
mvn versions:update-properties -Dincludes=com.hcl.products.test.it -f pomCustomSurefireSample.xml
```

- Run the test(s).

```
mvn clean verify -f pomCustomSurefireSample.xml
```

  Fail safe reports are generated in the target directory,especially in `target/failsafe-reports/<ProjectName>/<TestName>_<timestamp>.txt` that will contain the screen capture of the execution.

Method 2

With this method, no Maven report is generated. If you use your own pom.xml file, copy the following lines and provide your parameter values. Otherwise, you can use the `pomMojoExecPluginSample_Linux.xml` or `pomMojoExecPluginSample_Windows.xml` sample file.

Example with the `pomMojoExecPluginSample_Windows.xml` sample file:

- Copy `pomMojoExecPluginSample_Windows.xml` to a directory.
- Edit the file and update the arguments to reflect which test to execute. If the product is installed on a different drive or a different location, or if `IBMIMSharedHCLIMShared` location has been changed, update the two last lines with the path to the `IBMIMSharedHCLIMShared` plug-in folder.

```
<argument>/C</argument>
                                    <argument>${pt-plugin-cmdline}</argument>
                                    <argument>-workspace</argument>
                                    <argument>C:\Runtimes\runtime-RptMvn</argument>
                                    <argument>-project</argument>
                                    <argument>AA</argument>
                                    <argument>-suite</argument>
                                    <argument>Test1.testsuite</argument>
                                    <argument>-plugins</argument>
                                    <argument>C:/Program Files/IBM/IBMIMShared/pluginsC:/Program
 Files/HCL/HCLIMShared/plugins</argument>
```

- In the argument tags, instead of the -suite option, you can use the -aftsuite option and input the aft xml file as the parameter value in the subsequent argument tag to run the AFT test.

  For example, in the preceding template, `<argument>-suite</argument> <argument>Test1.testsuite</argument>` can be replaced with `<argument>-aftsuite</argument> <argument>aftfile.xml</argument>`.

- Run the test.

  For Windows

  ```
  mvn clean verify -f pomMojoExecPluginSample_Windows.xml
  ```

  For Linux or MacOS:

  ```
  mvn clean verify -f pomMojoExecPluginSample_Linux.xml
  ```

Related information

https://maven.apache.org/index.html

Supported options in Maven

Exemple

You must provide the values for the following required parameters:

- **workspace**
- **project**
- **suite**

If you include these required parameters in a configuration file and use the **configfile** parameter to specify the complete file path, then these parameters are not required.

Exemple

The following table lists all the supported options with description:

Option	Description
Required	
workspace	The complete path to the Eclipse workspace.
project	The path, including the file name of the project relative to the workspace.
suite	The path, including the file name of the test relative to the project. A test can be a Web UI test or a compound test.
Optional	
aftsuite	The path, including the file name of the xml file to run an AFT test. The **aftsuite** option accepts aft XML as the parameter value. It supports only one aft XML as input. For example, **aftsuite**="*aftinput*" **Note:** You must provide the file name along with the file extension to run an Accelerated Functional Test suite.
configfile	The option to specify the complete path to a file that contains the parameters for a test run. Each parameter must be on a single line. To create a configuration file, you must use an editor that does not wrap lines. Any parameter, whether required or optional, can be set in the configuration file. The command line parameters override the values in this file. **Notes:** • If you are creating a config file manually, the file must be in the UTF-8 format. You must not use quotation marks in this file even for values that contain spaces. • You can create command line config file from the desktop client, which you can use while running tests from Maven. See Creating a command-line config file on page 1400. This option is available only for Web UI and compound tests.
compare	The option to export the result in the compare mode. You must use this option with **exportstatshtml** and **execsummary**. The value can be paths to the runs and are relative to the workspace. You must separate the paths by a comma.
exportlog	The option to specify the file path to store the exported test log. You can provide multiple parameter entries when running multiple tests. You must use a colon to separate the parameter entries.

Option	Description
	For example: **exportlog** *c:/logexport.txt:c:/secondlogexport.txt* If there are multiple **suite** option entries with a single **exportlog** parameter entry, then the **exportlog** option generates the appropriate number of test logs by appending 0, 1, 2, and so on to the **exportlog** option entry name. For example: **suite**=*"sampletest1:sampletest2:sampletest3"* **exportlog**="*c:/logexport.txt* " The command generates the following test logs: • logexport_0.txt • logexport_1.txt • logexport.txt
exportstatreportlist	The option to specify a comma-separated list of report IDs along with **exportstats** or **exportstatshtml** to list the reports that you want to export in place of the default reports, or the reports selected under **Preferences**. To view this setting, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. To copy the report IDs list into your command line, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. Under **Select reports to export**, select the required reports, and click **Copy ID to clipboard**. You can then paste the clipboard content on to your editor.
exportstats	The option to export reports in comma-separated values (CSV) format, with the file name derived from the report name. This directory can be relative to the project or a directory on your file system. If the **exportstatreportlist** option is not specified, the reports specified on the **Export Reports** page of the **Performance Test Report** preferences are exported.
exportstatshtml	The option to export web analytic results to the specified directory. You can then analyze the results on a web browser without using the test workbench.
overwrite	Determines whether a result file with the same name is overwritten. The default value, `false`, indicates that the new result file is created. If the value is `true`, the file is overwritten and retains the same file name. You must use double quotes "" for values `true` or `false`.
plugins	The complete path to the folder that contains the plugins. Typically, on Windows operating systems, this folder is located at `C:\Program Files\IBM\IBMIMShared\pluginsC:\Program Files\HCL\HCLIMShared\plugins`. You must specify the path if the folder is at a different location.

Option	Description

protocolinput	The option to run a Web UI test in parallel on different browsers.
	
	`protocolinput="all.available.targets.in.parallel=all"`
	
	`protocolinput="all.available.targets.in.parallel=chrome,ff,ie"`
	
	Note: If you use the `protocolinput` argument, you must not use the equivalent vmargs arguments:
	`vmargs="-Dall.available.targets.in.parallel=all"`
	`vmargs "-Dall.available.targets.in.parallel=browser1,browser2,browser3"`

quiet	The option to turn off any message output from the launcher and return to the command shell when the run or the attempt is complete.
results	The name of the results file. The default result file name is the test name with a time stamp appended. You must specify a folder name that is relative to the project to store the test results. For example, **results**="*folder/resultname*"
usercomments	The text to be displayed in the **usercomments** row of the report. Add text within the double quotation mark (“”) **Note:** • When you run tests by using the double quotation marks (“”) for the **usercomments** field, then the **usercomments** row of a report does not contain double quotation marks. • To work around this problem, you must create a command-line config file, and then run the test by using the **configfile** field.
varfile	The complete path to the XML file that contains the variable name and value pairs. To run a Web UI test on a different browser than that was used for the recording, specify the predefined variable. For more information, see Defining a variable to run a test with a selected browser on page 578.
vmargs	The option to pass Java™ virtual machine arguments. You can use the **-vmargs** option in the following scenarios:

Option	Description
	• To specify the Java™ maximum heap size for the Java™ process that controls the command line playback, use the **-vmargs** option with the *-Xmx* argument. For example, when you use**-vmargs** *-Xmx4096m*, specify a maximum heap size of 4096m. This method is similar to specifying *-Xmx4096m* in the eclipse.ini file for the workbench when playing back the test from the user interface. • To collect the response time data for the app itself and for the server and network and display them in different bar charts, use **-vmargs** *"-De2e.collect=true"*. For desktop-based web applications, the response time data is captured and displayed by default. • To execute tests in parallel on all mobile devices, which are in passive mode, connected to the workbench and ready for playback, use **-vmargs** *"-Dall.available.targets.in-.parallel=true"*. • To execute tests in parallel on all supported desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=all"*. • To execute tests in parallel on selected desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=browser1,browswer2,browser3"*. You must separate browser names with a comma. For example, *firefox, ff, chrome, ie, ie64, safari, "-Dall.available.targets.in.parallel=browser1,browser2,browser3"*.
publish	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **publish** parameter: • To specify the project name, use any of the following formats: ◦ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _-teamspace* ◦ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _-teamspace_alias* You must consider the following points while providing the project name:

Option	Description

	◦ If the project name is not specified, then the value of the **Project** parameter is used.
	◦ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the_teamspace* or **&teamspace.alias=**=*name_of_the_teamspace_alias* options.
	For example:
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1"`
	Where:
	▪ https://localhost:5443 is the URL of the server.
	▪ test is the name of the project.
	▪ ts1 is the name of the team space.
	◦ While providing the name of the team space or team space alias, you must replace the ampersand (&) character with & as shown in the preceding example.
	◦ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`.
	For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*.
	Where, %20 is the hexadecimal value of the space character.
	• To avoid publishing of reports, use **no**.
	You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
	For example:
	`name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="no"`
	If you do not use the **configfile** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences

Option	Description
	(**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**). The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions: • When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**. • When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
publish_for	The option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **publish_for** parameter along with the **publish** parameter. The following are the available options that you can use for the **publish_for** parameter: • **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict. • **PASS** - You can use this option to publish test results for the tests that have passed. • **FAIL** - You can use this option to publish test results for the tests that have failed. • **ERROR** - You can use this option to publish test results for the tests that included errors. • **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests. You can add multiple parameters separated by a comma. For example: `name="test1" workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" suite="Tests/testHttp.testsuite" publish="https://localhost:5443#project.name=test&teamspace.name=ts1" publish_for="FAIL,ERROR"`
eclipsehome	The option to provide the complete path of the directory that contains `eclipse.exe`. For example, `C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest`

Option	Description
importzip	The option to import the project as test assets with dependencies into your workspace. For example, `workspace="C:/Users/IBM/rationalsdp/workspace1" project="proj1" eclipsehome="C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest" plugins="C:\Program Files\IBM\IBMIMShared\pluginsC:\Program Files\HCL\HCLIMShared\plugins" suite="Tests/testHttp.testsuite" importzip="D:\assets\test.zip"`
execsummary	The option to export all of the reports for the test run in a printable format, also known as an executive summary, to the local computer. You must specify the path to store the executive summary.
execsummaryreport	This option to export a specific report as an executive summary for the test run to the local computer. You must specify the ID of the report to export as **execsummaryreport <reportID>**. For example, to export an HTTP performance report, specify `http`. **Note:** You must use this option along with **execsummary**. To copy the report IDs list into your command line, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. Under **Select reports to export**, select the required reports, and click **Copy ID to clipboard**. You can then paste the clipboard content on to your command line editor.
swapdatasets	For a test, the default value is the dataset specified in the test editor. You must use the **swapdatasets** option to replace dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example: `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`. You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a semicolon (;).
exportReport	The option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML.

Option	Description
	Note: The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. The command syntax is as follows: exportReport "type=<reporttype>;format=<file type1,file type2,file type3>;folder<destination folder path;filename=<name of the exported file>" For example, to export the report to only the pdf format, you can use exportReport "type=unified;format=pdf;folder=Exportedreport102;filename=testreport" If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use exportReport "type=unified;format=pdf,xml,html;folder=Exportedreport102;filename=testreport" The report in different file formats use the same file name that is specified in the command.
labels	The option to add labels to the test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, **labels** =*"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use **labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:**

Option	Description
	• When you run tests by using the double quotation marks ("") for the **labels** parameter, then the labels in the test result do not include the double quotation marks. • To work around this problem, you must create a command-line config file, and then run the test by using the **configfile** parameter. • When you use the **configfile** parameter to run tests, then labels provided in the configuration file take precedence over the labels provided in the `Execute-WebUIFunctionalTest.xml` file.
exportstatsformat	The option to specify a format for the result that you want to export along with the **exportstats** option. You must use at least one of the following parameters with the **exportstatsformat** option: • simple.csv • full.csv • simple.json • full.json • csv • json For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*simple.json*" You can add multiple arguments separated by a comma. For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*simple.json*, *full.csv*" When you want to export both simple and full type of test results in a json or csv format, you can specify *json* or *csv* as the arguments in the command. When the test run completes, the test result exports to simple.json and full.json files. For example, **exportstats**="*<local_dir_path>*" **exportstatsformat**="*json*"

Exemple

Testing with Micro Focus ALM

You can integrate Rational® Functional TesterHCL OneTest™ UI with Micro Focus Application Lifecycle Management (ALM) to run test assets from Micro Focus ALM.

When you have Micro Focus ALM to manage the life cycle of your application under test, you can create test scripts to run it from Micro Focus ALM. You must use the content of the available template from the installation directory of Rational® Functional TesterHCL OneTest™ UI to create test scripts. The template is based on Microsoft VBScript and supports VAPI-XP test scripts. You can then run those test scripts from Micro Focus ALM and analyze the test results. For information about Micro Focus ALM, refer to ALM Help Center.

The following table lists the tasks that you must perform to run test assets from Micro Focus ALM:

Tasks	More information
Create Web UI tests in Rational® Functional TesterHCL OneTest™ UI to test your application.	See Creating Web UI tests on page 532.
Install Micro Focus ALM.	For more information about the installation of Micro Focus ALM, refer to the ALM Help Center. For more information about specific versions of software requirements, see Integration Middleware on page 30.
Create a test script in Micro Focus ALM.	See Creating a test script in Micro Focus ALM on page 405.
Copy the content of the template file and configure the test script in Micro Focus ALM.	See Configuring test scripts in Micro Focus ALM on page 406.
Run test assets as test scripts from Micro Focus ALM.	See Running tests from Micro Focus ALM on page 417.

Creating a test script in Micro Focus ALM

You must create a VAPI-XP-TEST type of test script on Micro Focus Application Lifecycle Management (ALM) to provide the details of the Web UI tests.

Before you begin

- You must be familiar with the Micro Focus ALM application.
- You must have performed the following tasks:
  - Installed Micro Focus ALM. For more information about the installation of Micro Focus ALM, refer to the ALM Help Center.
  - Been granted access to the Micro Focus ALM server.

1. Log in to the Micro Focus ALM portal, if you are not already logged in.
   The Micro Focus ALM dashboard is displayed.
2. Create a test by performing the following steps:

a. Expand **Testing** from the left pane, and then click **Test Plan**.

b. Select a folder from the available list to create a new test.

c. Click the **New Test** icon to create a new test.

d. Enter a name for the test in the **Test Name** field.

e. Select **VAPI-XP-TEST** as test type from the **Type** drop-down list, and then click **OK**.
Result
The **VAPI-XP Wizard** is displayed.

3. Select **VBScript** from the **Script Language** drop-down list.
4. Enter a name for the script in the **Script Name** field.

   The default name of the script is entered as *script*. You can change it by entering a new name.

5. Click **Next**, and then select **COM/DCOM Server Test** as a test type.
6. Click **Finish**.

Results
You have created the VAPI-XP-TEST test script in Micro Focus ALM.

What to do next
You must configure the test script to add the required parameter values of the Web UI test. See Configuring test scripts in Micro Focus ALM on page 406.

Configuring test scripts in Micro Focus ALM

You must configure the test script that you created in Micro Focus Application Lifecycle Management (ALM) to run the Web UI tests.

Before you begin

You must have performed the following tasks:

- Created a test script in Micro Focus ALM. See Creating a test script in Micro Focus ALM on page 405.
- Copied the content of the template file.

About this task

You can navigate to the `alm` directory to copy the content of the template file. The `alm` directory resides within the installation directory of Rational® Functional TesterHCL OneTest™ UI. The name of the template file is `WebUI_ALM_Windows.txt` and you can access the file from the following location:

`Installation_dir``\IBM\SDP\alm`*`Installation_dir`*`\HCL\HCLOneTest\alm`

For example, `C:\Program Files\IBM\SDP\almC:\Program Files\HCL\HCLOneTest\alm`

You must provide the values for the following required parameters:

- **Workspace**
- **Project**
- **TestSuiteName**

If you include these required parameters in a configuration file and use the **ConfigFile** parameter to specify the complete file path, then these parameters are not required.

Important:

You must enter the parameter values within the double quotation marks. If the values of the parameter contain the double quotation marks, then the values must be enclosed in another double quotation marks. For example, if you want to add a label for a test result as `"perf mon"`, then you must enter the parameter value in the script as follows:

```
Labels = ""perf mon""
```

1. Log in to the Micro Focus ALM portal, if you are not already logged in.
   The Micro Focus ALM dashboard is displayed.
2. Expand **Testing** from the left pane, and then click **Test Plan**.
3. Select a test script from the folder that you want to configure.
4. Click the **Test Script** tab.
5. Paste the content of the `WebUI_ALM_Windows.txt` file that you copied in the space provided in the **Test Script** tab.
6. Enter the parameter values that are required for your test run in the script by referring to the following table:

Parameter	**Description**
Required	
Workspace	Use this parameter to enter the complete path of the Eclipse workspace.
Project	Use this parameter to enter the name of the project that has test assets.
TestSuiteName	Use this parameter to enter the name of the test assets. For example, `Workspace = "C:/Users/IBM/rationalsdp/workspace1C:\Users\HCL\hclonetest\workspace1"`

Parameter	Description
	`Project = "proj1"` `TestSuiteName = "Tests/testHttp.testsuite"`
Optional	
ConfigFile	Use this parameter to provide the complete path to a configuration file that contains the parameters for a test run.
ExportReport	Use this option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. The command syntax is as follows: exportReport "type=<reporttype>;format=<file type1,file type2,file type3>;folder<destination folder path>;filename=<name of the exported file> For example, to export the report to only the pdf format, you can use exportReport "type=unified;format=pdf;folder=Exportedreport102;filename=testreport If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use exportReport "type=unified;format=pdf,xml,html;folder=Exportedreport102;filename=testreport The report in different file formats use the same file name that is specified in the command.
ExportStatReportlist	Use this parameter to provide a comma-separated list of absolute paths to custom report format files (.view files) that you can use to export statistical report data with **ExportStatsFile**.

Parameter	Description
ExportStatsFile	Use this parameter to provide the complete path to a directory that you can use to store exported statistical report data.
ExportStatsHtml	Use this parameter to provide the complete path to a directory that you can use to export web analytic results. The results are exported to the specified directory. You can analyze the results on a web browser without opening Rational® Functional TesterHCL OneTest™ UI.
IMSharedLocation	Use this parameter to enter the complete path to the IMSharedLocationHCLIMShared location.

Labels	Use this parameter to add labels to test results when the test run is complete. You can add multiple labels to a test result separated by a comma (,).
	
	For example,
	
	`Workspace = "C:/Users/IBM/rationalsdp/workspace1C:\Users\HCL\hclonetest\workspace1"`
	`Project = "proj1"`
	`TestSuiteName = "Tests/testHttp.testsuite"`
	`Labels = "label1, label2"`
	
	When you run test assets, then the labels that you added are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI.
	
	The **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the label that you added in the **Labels** parameter for the specific test asset in the following condition:
	
	◦ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Windows > Preferences > Test > Rational Test Automation Server > HCL OneTest Server > Results**).
	
	Note:

Parameter	Description
	◦ When you run tests by using the double quotation marks ("") for the **Labels** parameter, then the labels in the test result do not include the double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **ConfigFile** parameter. ◦ When you use the **ConfigFile** parameter to run tests, then labels provided in the configuration file take precedence over the labels provided in the test script.
OverwriteResultsFile	Set this parameter value to `true` or `false` to determine whether a result file with the same name must be overwritten or not. The default value is `true`.
Publish	Use this option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server from Microfocus ALM. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **Publish** parameter in the test script. Use the following arguments with the **Publish** parameter:

Parameter	Description
	◦ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias* You must consider the following points while providing the project name: ▪ If the project name is not specified, then the value of the **Project** parameter is used. ▪ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias=**=*name_of_the_teamspace_alias* options. For example: `Workspace = "C:/Users/IBM/rationalsdp/workspace1" Project = "proj1" TestSuiteName = "Tests/testHttp.testsuite" Publish = "https://localhost:5443#project.name=test&teamspace.name=ts1"` Where: ▪ https://localhost:5443 is the URL of the server. ▪ test is the name of the project. ▪ ts1 is the name of the team space.

Parameter	Description

	▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_-value_of_special_character>`.
	
	For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team %20Space*.
	
	Where, %20 is the hexadecimal value of the space character.
	
	◦ To avoid publishing of reports, use **no**.
	
	You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
	
	For example:
	
	`Workspace = "C:/Users/IBM/rationalsdp/workspace1"`
	`Project = "proj1"`
	`TestSuiteName = "Tests/testHttp.testsuite"`
	`Publish = "no"`
	
	If you do not use the **Config File** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
	
	The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:
	
	◦ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Prefer-**

Parameter	Description
	ences > Test > Rational Test Automation Server > HCL One Test Server. ◦ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).

Publish_for	Use this parameter to publish test results to Rational® Test Automation ServerHCL OneTest™ Server from Micro Focus ALM based on the completion status of the tests. You must use the **Publish_for** parameter along with the **Publish** parameter.
	The following are the available options that you can use for the **Publish_for** parameter:
	◦ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict.
	◦ **PASS** - You can use this option to publish test results for the tests that have passed.
	◦ **FAIL** - You can use this option to publish test results for the tests that have failed.
	◦ **ERROR** - You can use this option to publish test results for the tests that included errors.
	◦ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.
	You can add multiple parameters separated by a comma.
	For example:
	`Workspace = "C:/Users/IBM/rationalsdp/workspace1"`
	`Project = "proj1"`
	`TestSuiteName = "Tests/testHttp.testsuite"`
	`Publish = "https://localhost:5443#project.name=test&teamspace.name=ts1"`
	`Publish_for = "FAIL,ERROR"`

Parameter	Description

Publishreports	Use this option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **Publishreports** are as follows:
	◦ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455.
	◦ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.
	◦ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
	
	You must use the **Publishreports** parameter along with the **Publish** parameter.
	
	For example:
	`Workspace = "C:/Users/IBM/rationalsdp/workspace1" Project = "proj1" TestSuiteName = "Tests/testHttp.testsuite" Publish = "https://localhost:5443#project.name=test&teamspace.name=ts1" Publishreports = "STATS, TESTLOG"`
	
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	
	For example,
	`Workspace = "C:/Users/IBM/rationalsdp/workspace1" Project = "proj1" TestSuiteName = "Tests/testHttp.testsuite"`

Parameter	Description
	``` Publish = "https://localhost:5443#project.name=test&teamspace.name=ts1" Publishreports = "!TESTLOG" ``` All the reports except the TESTLOG report is published to Rational® Test Automation ServerHCL OneTest™ Server after executing the command.
ProtocolInput	Use this option with additional arguments as follows: ◦ To run a Web UI test in parallel on different browsers: **ProtocolInput** = *"all.available.targets.in.parallel=all"* **ProtocolInput** = *"all.available.targets.in.parallel=chrome,ff,ie"* **Note:** If you use the **ProtocolInput** option, you must not use the following equivalent **VMArgs** arguments: ``` VMArgs "-Dall.available.targets.in.parallel=all" VMArgs "-Dall.available.targets.in.parallel=browser1,browswer2,browser3" ``` ◦ To specify the Web UI preferences such as highlighting the page element and capturing screens: For example, **ProtocolInput** = *"webui.highlight=<value>;webui.report.screenshots=<value>"* where *webui.highlight* specifies whether the page element must be highlighted and *webui.report* specifies whether the screens must be captured while playing back the test in the browser.

Parameter	Description
Quiet	Use this parameter to turn off any message output from the launcher and return to the command shell when the run or the attempt is complete.
ResultsFile	Use this parameter to provide a different name to the result file. The default name of the result file is the name of the test or schedule with a timestamp appended.
UserComments	Use this parameter to display comments that you added in the **UserComments** row of a report. **Note:** ◦ When you run tests by using the double quotation marks ("") for the **UserComments** parameter, then the **UserComments** row of a report does not contain double quotation marks. ◦ To work around this problem, you must create a command-line config file, and then run the test by using the **ConfigFile** parameter.
VarFile	Use this parameter to provide a complete path to the XML file that contains the variable name and value pairs.
VMArgs	Use this parameter to pass Java™ virtual machine arguments.

7. Click the **Save** icon.

Results

You have configured the test script by adding the required parameter values for the test run.

What to do next

You can run test assets from Micro Focus ALM. See Running tests from Micro Focus ALM on page 417.

Running tests from Micro Focus ALM

You can run test assets from Micro Focus Application Lifecycle Management (ALM) to run the Web UI tests in Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Created a test script in Micro Focus ALM. See Creating a test script in Micro Focus ALM on page 405.
- Configured the test script in Micro Focus ALM. See Configuring test scripts in Micro Focus ALM on page 406.
- **Optional**: Generated offline user token to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. For more information refer to Managing access to the serverManaging access to the server in the Rational® Test Automation ServerHCL OneTest™ Server Documentation.

About this task

After the test run is complete, you can access the report to view the test run information. The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:

- When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Windows > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**).
- When you set **Publish result after execution** to **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Windows > Preferences > Test > Rational Test Automation Server > HCL OneTest Server > Results**).

1. Log in to the Micro Focus ALM portal, if you are not already logged in.
   The Micro Focus ALM dashboard is displayed.
2. Expand **Testing** from the left pane, and then click **Test Plan**.
3. Open the test script that you want to run.
4. Click the **Test Script** tab.
   Result
   The test script is displayed.
5. Click the **Execute Script** icon to run the test script.

Results

You have run the Web UI test scripts from Micro Focus ALM.

What to do next

You can view the test result details that are displayed in the **Output** window of Micro Focus ALM.

Testing with Rational® Integration TesterHCL OneTest™ API

You can use Rational® Integration TesterHCL OneTest™ API extension to run integrationAPI tests from Rational® Functional TesterHCL OneTest™ UI. You can either import the projects from Rational® Integration TesterHCL OneTest™ API or manage them from Rational® Functional TesterHCL OneTest™ UI by establishing the connection between the products. You can also create a compound test to run the tests by using the Agents.

Before you begin

You must have completed the following tasks:

- Installed Rational® Integration TesterHCL OneTest™ API Extension for Rational® Integration TesterHCL OneTest™ API.
- Installed Rational® Performance TesterHCL OneTest™ Performance Agent and Rational® Integration Tester AgentHCL OneTest™ API Agent to run tests remotely. When you install Rational® Integration TesterHCL OneTest™ API, you must have selected the **This Agent will only run probes** option.
- Defined the environment variable *INTEGRATION_TESTER_AGENT_HOME* on each location where the Agent is installed, and have pointed to the root installation directory of Rational® Integration Tester AgentHCL OneTest™ API Agent.

Note: The environment variables *INTEGRATION_TESTER_HOME* and *INTEGRATION_TESTER_AGENT_HOME* are automatically created when you install Rational® Integration TesterHCL OneTest™ API 10.2.2 or later.

Setting environment variable

You must set the environment variable *INTEGRATION_TESTER_AGENT_HOME* and point it to the Rational® Integration Tester AgentHCL OneTest™ API Agent installation directory if you are using Rational® Integration Tester AgentHCL OneTest™ API Agent 10.2.1 or earlier.

1. Run the following command to set the environment variable on Windows:

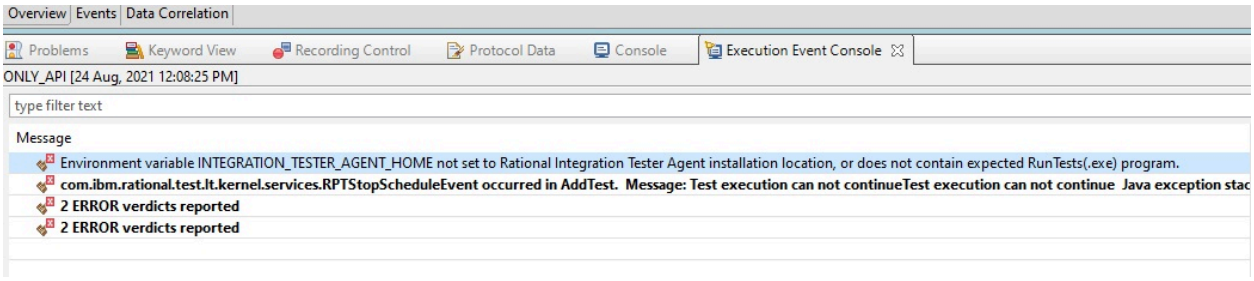
```
INTEGRATION_TESTER_AGENT_HOME = C:\Program Files\HCL\HCLProducts\Agent
```

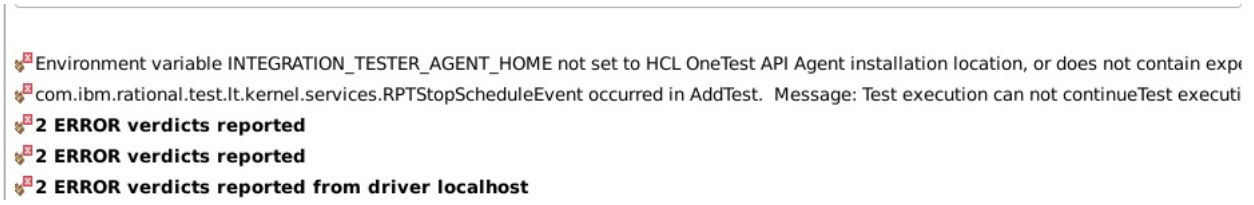
```
INTEGRATION_TESTER_AGENT_HOME = C:\Program Files\IBM\Agent
```

2. Run the following commands to set the environment variable on Linux:

```
INTEGRATION_TESTER_AGENT_HOME=/opt/IBM/RIT-Agent/opt/HCL/HOT-API-Agent
export INTEGRATION_TESTER_AGENT_HOME
echo $INTEGRATION_TESTER_AGENT_HOME
```

If the environment variable is not set, the compound or schedule test run fails with an error message as depicted in the following image:

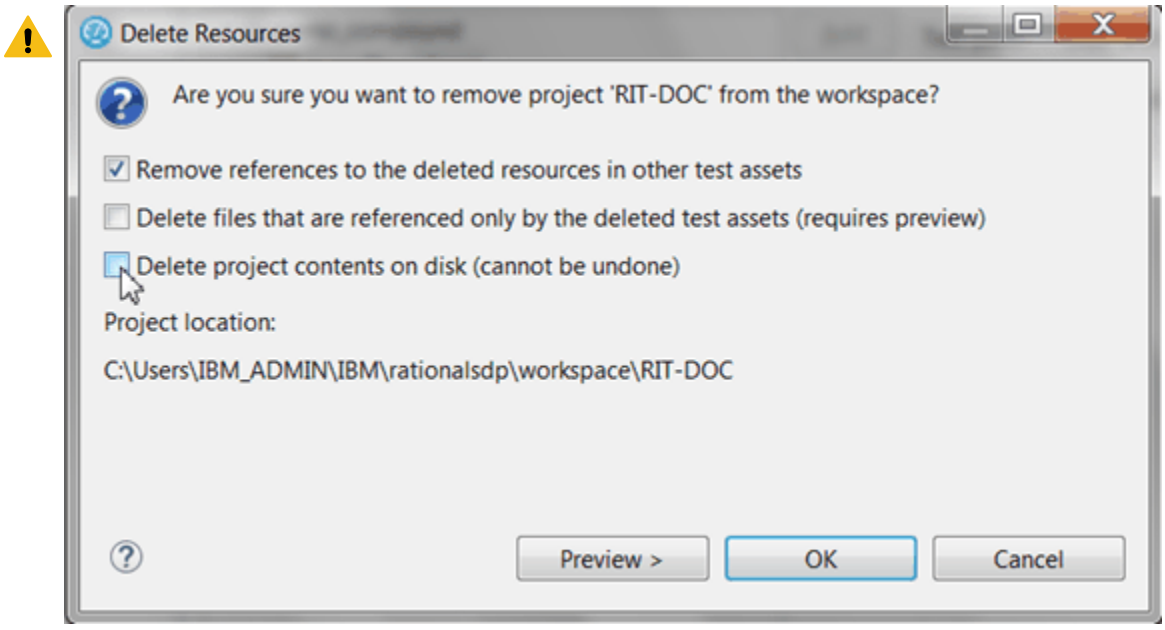




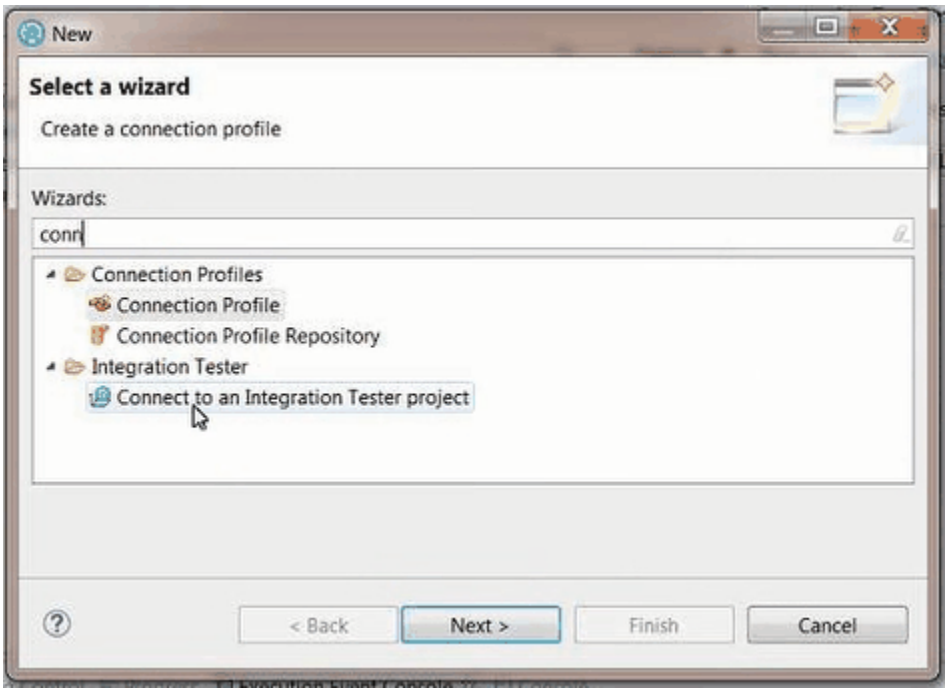
Connecting to an existing IntegrationAPI project

When you connect both the products any change or delete action made in one product workspace is reflected on the other product workspace, if both the products are installed on your machine.

Warning: If you delete a project from the Test Navigator, be sure that the option **Delete project contents on disk** is not selected in the **Delete Resources** dialog, otherwise the project would be deleted in Rational® Integration TesterHCL OneTest™ API if it is connected.



- In Rational® Functional TesterHCL OneTest™ UI, right-click on the **Test Navigator**, select **New > Other > Rational Integration Tester > HCL OneTest API > Connect to an IBM Rational Integration Tester Project > Connect to an HCL OneTest API Project** and click **Next**.



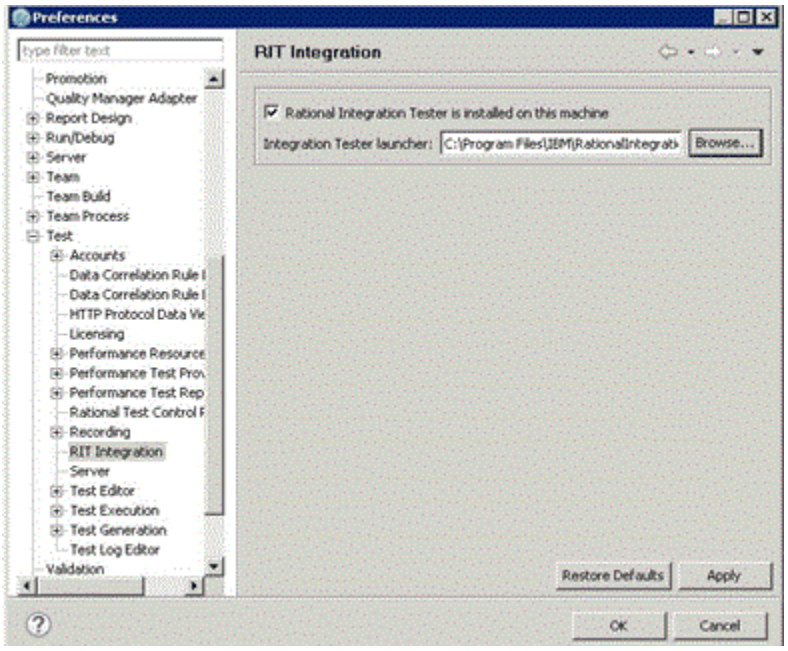
- In the wizard page, click **Browse** and select the root folder that contains the project.

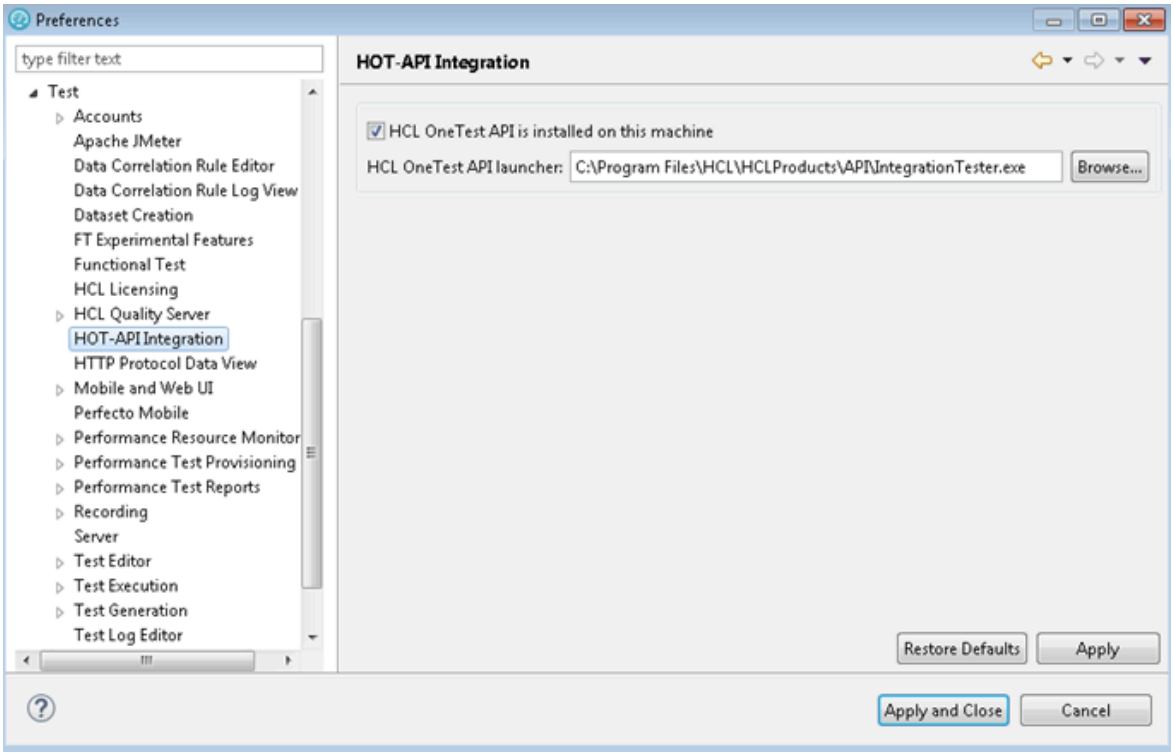
  If the path contains a project, its name should automatically appear in **Project Name** and the **Finish** button should be enabled.

Setting Rational® Integration TesterHCL OneTest™ API preferences

To be able to open an Rational® Integration TesterHCL OneTest™ API project from Rational® Functional TesterHCL OneTest™ UI Test Navigator, you need to have both the products installed on the same computer, and you must set the path to the execution file in the Preferences.

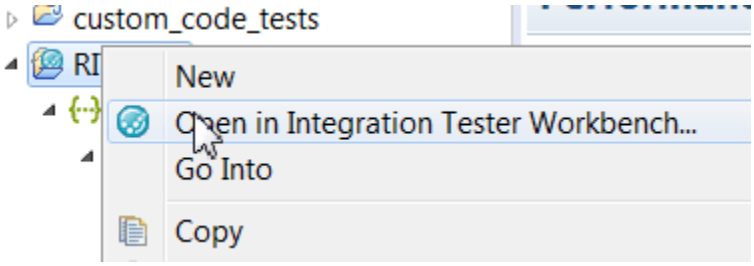
- In Rational® Functional TesterHCL OneTest™ UI, click **Window > Preferences > Test > RIT Integration > HIT Integration**.
- Click **Browse** and set the installation path to Rational® Integration TesterHCL OneTest™ API execution file. On Windows, the default location can be `C:\Program Files\IBM\IntegrationTester.exeC:\Program Files\HCL\IntegrationTester.exe`.
- Click **Apply** and **OK**.





Opening Rational® Integration TesterHCL OneTest™ API resources from the Test Navigator

- Once the preferences are set, you can open an Rational® Integration TesterHCL OneTest™ API project.
- In the **Test Navigator**, open the project root node and children nodes, and at any level, right-click and select **Open in Rational Integration Tester WorkbenchOpen in HCL OneTest API Workbench**.



If Rational® Integration TesterHCL OneTest™ API is automatically detected, the workspace opens for the selected resources.

If Rational® Integration TesterHCL OneTest™ API is not detected, a dialog opens on a Preference page where you need to verify the path to the execution file.

- **Warning:** Rational® Integration TesterHCL OneTest™ API cannot open more than one project at a time. If you have another project open, an error is displayed. In that case, close Rational® Integration TesterHCL OneTest™ API and try to open the project again.

Importing Rational® Integration TesterHCL OneTest™ API project

If both the products are not installed on the same machine, you can import an Rational® Integration TesterHCL OneTest™ API project in your workspace. Another reason for the import is when you have Rational® Integration TesterHCL OneTest™ API installed but you do not want to connect to the Rational® Integration TesterHCL OneTest™ API project. In that case, the project is duplicated, any updates in one product workspace is not reflected in the workspace of the other product.

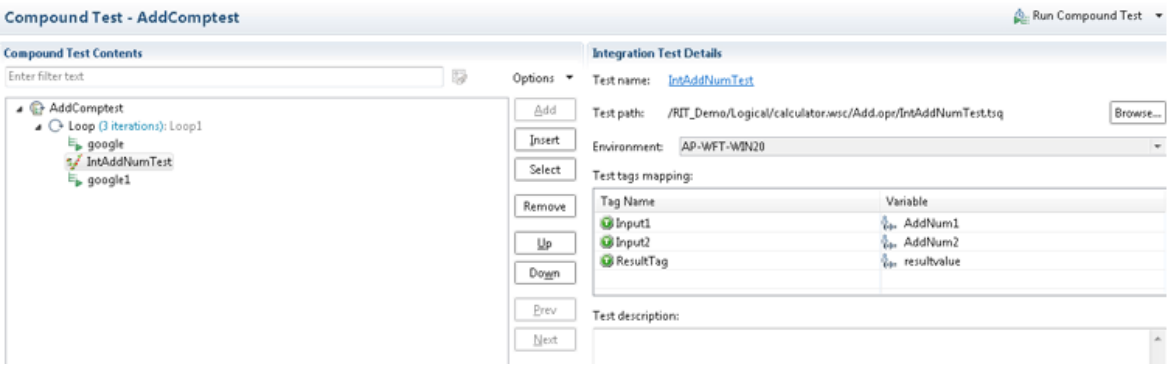
- To import an Rational® Integration TesterHCL OneTest™ API project:
- Right-click on the **Test Navigator**, choose **Import** and select **Existing project into workspace**.
- Choose **Select root directory** or **Select archive file**; select a project to import and click **Finish**.

The selected project appears in the **Test Navigator** and the compound test editor automatically opens.

Modifying Rational® Integration TesterHCL OneTest™ API environments in UI Test perspective

In the compound test, you can select Rational® Integration TesterHCL OneTest™ API tests and change the environment of each test. The environments are set in Rational® Integration TesterHCL OneTest™ API, you can only change the selection from the edited compound test.

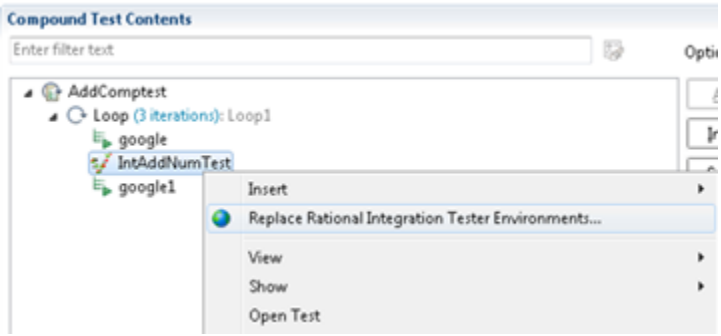
- Open the compound editor and select a test.
- In the Rational® Integration TesterHCL OneTest™ API details, you can browse and change the properties of the selected test. The **Test path**, the **Environments** and **Description** are automatically updated accordingly.



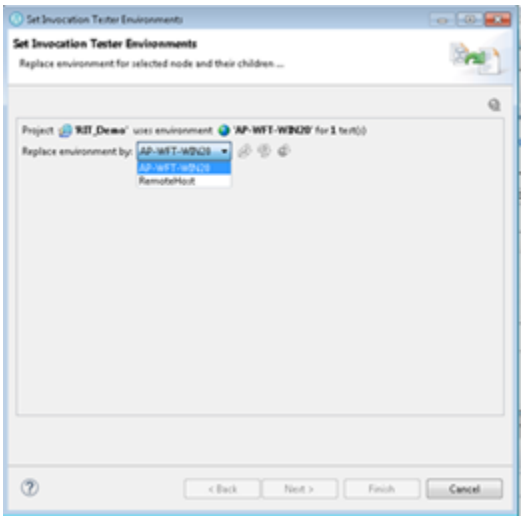
- To select another environment for the Integration Tester test, use the dropdown menu.

Alternatively, you can change the environment selection for a test for a collection of tests:

- Right-click on the tree at any level under a node in the compound test and select **Replace Rational Integration Tester EnvironmentsReplace HCL OneTest API Environments**.



- In the **Set Invocation Tester Environments** wizard, the first page displays the list of projects that use the selected environment and the number of tests from project that use this environment in the compound test.



- Select another used environment in the dropdown list. Click **Finish**. The new choice applies to the selected node and its children.

Next step is to create a compound test in Rational® Functional TesterHCL OneTest™ UI to run the test. See Running Rational Integration TesterHCL OneTest API tests on page 426.

To map the tags in Rational® Integration TesterHCL OneTest™ API with variables in Rational® Functional TesterHCL OneTest™ UI, you must follow these steps:

1. Click the Rational® Integration TesterHCL OneTest™ API test that contains the tags that you want to map.

   The Rational® Integration TesterHCL OneTest™ API tags are displayed in the **Integration Test Details** pane.
2. Click the tag that you want to map with the variable in Rational® Functional TesterHCL OneTest™ UI.
3. Click the Ellipses button inline with the tag that you want to map with a variable.

   The **Variable Selection** dialog box is displayed.
4. Select the variable, and then click **OK**.

   The tag is mapped to the selected variable and it is displayed in the **Variable** column.

Note: While you map a tag in Rational® Integration TesterHCL OneTest™ API test to a variable in Rational® Functional TesterHCL OneTest™ UI, you can map only tags that contain the string value to variables. You cannot map a tag that contains multiple values to a variable in Rational® Functional TesterHCL OneTest™ UI.

You can add a dataset mapper in the compound test for tests that are using multiple tags. See Adding Dataset Mapper on page 645 to map tags in the Rational® Integration TesterHCL OneTest™ API tests with the variable values of Rational® Functional TesterHCL OneTest™ UI.

Running Rational® Integration TesterHCL OneTest™ API tests

You can use Rational® Functional TesterHCL OneTest™ UI Extension for Rational® Integration TesterHCL OneTest™ API to run integrationAPI tests.

You also have the option to just import the projects to Rational® Functional TesterHCL OneTest™ UI from Rational® Integration TesterHCL OneTest™ API, add the tests to a compound test to run them. You can either use Rational® Performance TesterHCL OneTest™ Performance Agent or Rational® Integration TesterHCL OneTest™ API Agent to generate the load. You need a compound test that contains the Rational® Integration TesterHCL OneTest™ API tests.

If you update integrationAPI tests in Rational® Functional TesterHCL OneTest™ UI and want to apply the updates back to Rational® Integration TesterHCL OneTest™ API, you must install Rational® Integration TesterHCL OneTest™ API and define the path to its installation directory to set the connection.

You can find the following use cases to work with integrationAPI tests in Rational® Functional TesterHCL OneTest™ UI:

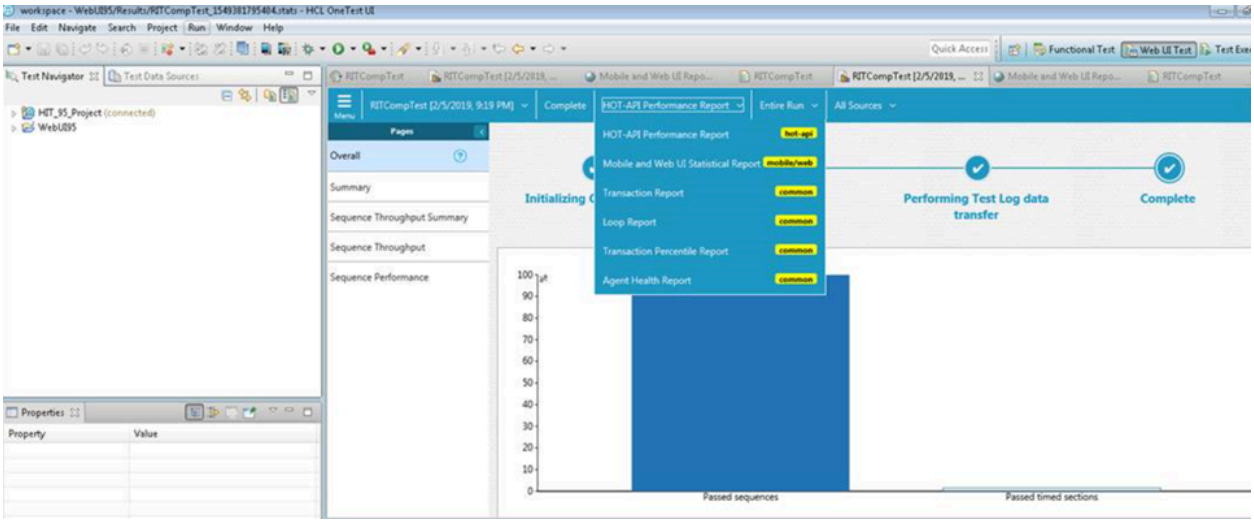
- You have installed both the products and then you connect to the IntegrationAPI project. Alternatively, you open the IntegrationAPI resource directly from the **Test Navigator** view, and then work directly with the sources files.
- Rational® Integration TesterHCL OneTest™ API is not installed and you import the projects in the workspace of Rational® Functional TesterHCL OneTest™ UI.

  Note: The imported tests must be edited in Rational® Integration TesterHCL OneTest™ API. Similarly, the compounds tests must be edited in Rational® Functional TesterHCL OneTest™ UI.

Running the compound test

Click **Run Compound Test**.

After the run completes, the Rational® Integration TesterHCL OneTest™ API report displays statistics on the executed sequences and Timed Sections if some are defined in the tests.



Testing with HCL® Launch

You can integrate Rational® Functional TesterHCL OneTest™ UI with HCL® Launch to run tests as a **Process** from HCL® Launch.

When you use HCL® Launch for automating the deployments of your applications to an environment during the development cycle, you can create tests for your application in Rational® Functional TesterHCL OneTest™ UI. You can then run those tests from HCL® Launch.

You must use the HCL® Launch plugin to integrate Rational® Functional TesterHCL OneTest™ UI with HCL® Launch. Integrating Rational® Functional TesterHCL OneTest™ UI with HCL® Launch automates the process of test runs. If you have many tests to run at regular intervals, you can use HCL® Launch to initiate test runs automatically.

The following table lists the tasks that you must perform to run tests from HCL® Launch:

Tasks	Go to...
Create any or all the following types of tests in Rational® Functional TesterHCL OneTest™ UI to test your application: • Web UI tests • Compound tests	Test Author Guide on page 532.
Install HCL® Launch with at least one agent.	HCL® Launch documentation. System requirements on page 28 for information about specific versions of HCL® Launch requirements.
Install the Rational® Functional TesterHCL OneTest™ UI Launch plugin on HCL® Launch.	Installing the plugin on HCL Launch on page 428.

Tasks	Go to...
Create a **Component** in HCL® Launch.	Creating a component in HCL Launch on page 429.
Create a **Process** for the component in HCL® Launch.	Creating a process in HCL Launch on page 429.
Configure the **Process** that you created for the component in HCL® Launch.	Configuring the process on page 430.
Create a **Resource** in HCL® Launch.	Creating a resource in HCL Launch on page 442.
Configure the **Resource** in HCL® Launch.	Configuring the resource on page 443.
Create an **Application** in HCL® Launch.	Creating an application in HCL Launch on page 444.
Configure the **Application** that you use to run tests in HCL® Launch.	Configuring the application on page 445.
Run the tests from HCL® Launch.	Running tests from HCL Launch on page 446.

Installing the plugin on HCL® Launch

You must install the Rational® Functional TesterHCL OneTest™ UI Launch plugin to run tests from HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Installed HCL® Launch. For more information about the installation, refer to the HCL® Launch documentation.
  - Been granted access to HCL® Launch.
  - Downloaded the Rational® Functional TesterHCL OneTest™ UI Launch plugin `HOT-UI-LAUNCH-UITest-10.1` from the HCL® License & Delivery portal.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Settings**.
3. Click **Automation Plugins** from the **Automation** pane.
4. Click **Load Plugin**.
5. Click **Choose File** to locate and open the compressed Rational® Functional TesterHCL OneTest™ UI Launch plugin file.

   Remember: You must not extract the Rational® Functional TesterHCL OneTest™ UI Launch plugin compressed file contents.

6. Click **Submit**.
   The installed plugin is displayed on the **Automation Plugins** tab.

Results

You have installed the Rational® Functional TesterHCL OneTest™ UI Launch plugin on HCL® Launch.

What to do next

You must create a **Component** in HCL® Launch. See Creating a component in HCL Launch on page 429.

Creating a component in HCL® Launch

You must create a component to include artifacts and processes. The artifacts include runnable files, images, databases, configuration instructions. Whereas the processes define the activities that components can perform.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**, and then click **Create Component**.
3. Enter a name for the component in the **Name** field.
4. Enter the details in the other optional fields based on your requirement, and then click **Save**.
   The component that you created is displayed.

Results

You have created the component in HCL® Launch.

What to do next

You must create a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 429.

Related information

HCL Launch Documentation

Creating a process in HCL® Launch

You must create a process for the component to include step properties for the test that you want to run from HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 429.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**.
   A list of components that are available in HCL® Launch is displayed.
3. Select the component from the list for which you want to create a process.
4. Click the **Processes** tab, and then click **Create Process**.
   The **Create Process** dialog is displayed.
5. Enter a name for the process in the **Name** field.
6. Select **Operational (No Version Needed)** from **Process Type** drop-down list.
7. Verify the **Default Working Directory** field.

   The **Default Working Directory** field defines the location that the agent uses to run the process. The default value is `${p:resource/work.dir}/${p:component.name}`.

   Where `${p:resource/work.dir}` is the default working directory for the agent and `${p:component.name}` is the name of the component.
8. Click **Save**.
   The process that you created is listed in the **Processes** tab and the **Design** tab for the process is displayed.

Results

You have created the process for the component in HCL® Launch.

What to do next

You must configure the process in HCL® Launch. See Configuring the process on page 430.

Related information

HCL Launch Documentation

Configuring the process

You must configure the process that you created for the component to organize the steps in the process, specify the properties of the steps, and connect them.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 429.
  - Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 429.

About this task

When you open any process to configure, the process is displayed in the process editor. The process editor lists the plugins and steps. The required **Start** and **Finish** steps represent the beginning and the end of the process and are automatically placed on the design area.

You must provide the values for certain fields in the properties for the selected test step to run tests from HCL® Launch. The minimum fields that you need to configure are **Name**, **Workspace**, **Project**, and **Test Suite Name** or **Config File**. The remaining fields are optional.

For example, if you provide a complete path of a configuration file that contains the parameters for a `test` run in the **Config File** field, then you need not provide values for the **Workspace**, **Project**, and **Test Suite Name** fields.

Note: If Rational® Functional TesterHCL OneTest™ UI is not installed in default location, then you must provide the value for the **IMShared Location** field along with other mandatory fields.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**.
   A list of components that are available in UrbanCode™ Deploy is displayed.
3. Select the component from the list in which you created the process.
4. Click the **Processes** tab.
   A list of processes that are available for the component is displayed.
5. Select the process from the list that you want to configure.
   The **Design** tab for the process is displayed.
6. Click **HCL OneTest Studio**, and then **HCL OneTest UI** from the left menu.
7. Drag the **Run an HCL OneTest UI - UI test** step, and then drop it into the design area.

   Note: The selected test must be placed between **Start** and **Finish** steps.

8. Specify the properties for the selected test by performing the following steps:

   a. Click the **Edit** icon.
      Result
      The **Edit Properties for Run an HCL OneTest UI - UI Test** dialog is displayed.

   b. Specify the properties for the selected test step by referring to the following table:

Field	Description
Required	
Name	Enter the name of the step.
Workspace	Enter the complete path to the Eclipse work-space.

Field	Description
Project	Enter the path, including the file name of the project relative to workspace.
Test Suite Name	Enter the path, including the file name, of the test to run relative to the project. A test can be a Web UI test, compound test, or Accelerated Functional Test. To run multiple tests sequentially, specify test names separated by comma.
Optional	
IMShared Location	Enter the complete path to the location of the IBMIMSharedHCLIMShared folder. For example, `C:\Program Files\IBM\IBMIMSharedC:\Program Files\HCL\HCLIMShared`
Import	Enter the complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets in the Workspace field.
Var File	Enter the complete path to the XML file that contains the variable name and value pairs.
Config File	Enter the complete path to a file that contains the parameters for a test run.
Results File	Enter the name of the results file. The default result file is the test name with a time stamp appended.
Overwrite Results file	Determines whether a results file with the same name is overwritten. The default value is true which indicates that the results files can be overwritten.
Number of Virtual Users	For a schedule, the default value is the number of users specified in the schedule editor. For a test, the default value is one user. Overrides the default number of users, if required.
VM Args	Java™ virtual machine arguments to pass in.
Protocol Input	Use this argument to run a Web UI test in parallel on different browsers.

Field	Description

	`-protocolinput "all.available.targets.in.parallel=all"`
	
	`-protocolinput "all.available.targets.in.parallel=chrome,ff,ie"`
	
	Note: If you use the `-protocolinput` argument, you must not use the equivalent -vmargs arguments:
	`-vmargs "-Dall.available.targets.in.parallel=all" -vmargs "-Dall.available.targets.in.parallel=browser1,browser2,browser3"`

Dataset Override	For a test, the default value is the dataset specified in the test editor. **Note:** You must use the `Dataset Override` option to replace the dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example, `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`. You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a `semicolon (;)`. For example,

Field	Description
	/project_name1/ds_path/ds_filename.csv:/project_name1/ds_path/new_ds_filename.csv; /project_name2/ds_path/ds_filename.csv:/project_name2/ds_path/new_ds_filename.csv
Exported HTTP Test log File	Enter the complete path to a file in which to store the exported HTTP test log.
Exported Statistical Report Data File	Enter the complete path to the directory in which to store exported statistical report data.
Custom Report Format Files	A comma-separated list of absolute paths to custom report format files (.view files) to use when exporting statistical report data with the -exportstats option.
User Comments	Enter the text within double quotation mark to display it in the **User Comments** row of the report. **Note:** ▪ When you run tests by using the double quotation marks ("") for the **User Comments** field, then the **User Comments** row of a report does not contain double quotation marks. ▪ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field.
Export Report	Specify the values to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. Enter the following details in the text field: *type=<report type>;format=<file type>;folder=<destination folder path>;filename=<name of the exported file>*

Field	Description
	For example, type=unified;format=pdf,xml,html;folder=E:\temp\logs;filename=UCDExecutionReport When you choose to export the report to all the supported formats, the report in different file formats use the same file name that is specified in the field.
Exportstatsformat	Specify a format for the report that you want to export. You must use the **Exported Statistical Report Data File** parameter along with the **Export stats format** parameter. You must use at least one of the following formats: ▪ *simple.csv* ▪ *full.csv* ▪ *simple.json* ▪ *full.json* ▪ *csv* ▪ *json* For example, **Exported Statistical Report Data File** = *<local_dir_path>* **Export stats format** = *json* You can add multiple formats for the report separated by a comma (,). If you want to export both the simple and full reports in a *json* or *csv* format, you can specify *json* or *csv* as the format in the field. The reports are saved to the location specified in the **Exported Statistical Report Data File** field.

Field	Description
Labels	Enter the labels to be added to the test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, *"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:** ▪ When you run tests by using the double quotation marks ("") for the **Labels** field, then the labels in the test result do not include the double quotation marks. ▪ To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field. ▪ When you use the **Config File** field to run tests, then labels provided in the configuration file take precedence over the labels provided in the **Labels** field.
Exportstatshtml	Enter the complete path to a directory to export web analytic results. The results are exported to the specified directory. You can then analyze the results on a web browser without using the test workbench.

Field	Description
Publish	Specify the server URL and project name to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **Publish** parameter: ▪ To specify the project name, use any of the following formats: ▪ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ▪ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias* You must consider the following points while providing the project name: ▪ If the project name is not specified, then the value of the **Project** parameter is used. ▪ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias==***name_of_the_teamspace_alias* options. For example:

Field	Description
	``` Workspace = C:/Users/IBM/rationalsdp/workspace1 Project = proj1 Test Suite Name = Tests/testHttp.testsuite Publish = https://localhost:5443#project.name=test&teamspace.name=ts1 ``` Where: ▪ https://localhost:5443 is the URL of the server. ▪ test is the name of the project. ▪ ts1 is the name of the team space. ▪ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`. For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Intial%20Team%20Space*. Where, %20 is the hexadecimal value of the space character. ▪ To avoid publishing of reports, use **no**. You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them. For example:

Field	Description
	``` Workspace = C:/Users/IBM/rationalsdp/workspa ce1 Project = proj1 Test Suite Name = Tests/testHttp.testsuite Publish = no ``` If you do not use the **Config File** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**). The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions: ▪ When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**. ▪ When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
Publish_for	Specify the option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **Publish_for** parameter along with the **Publish** parameter. The following are the available options that you can use for the **Publish_for** parameter:

Field	Description

	▪ **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict.
	▪ **PASS** - You can use this option to publish test results for the tests that have passed.
	▪ **FAIL** - You can use this option to publish test results for the tests that have failed.
	▪ **ERROR** - You can use this option to publish test results for the tests that included errors.
	▪ **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.
	You can add multiple parameters separated by a comma.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish_for = FAIL,ERROR`

Publishreports	Specify the option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **Publishreports** are as follows: ▪ FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455. ▪ STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.

Field	Description

	▪ TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
	
	You must use the **Publishreports** parameter along with the **Publish** parameter.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publishreports = STATS, TESTLOG`
	
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	
	For example,
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publishreports = !TESTLOG`
	
	All the reports except the TESTLOG report is published to Rational® Test Automation ServerHCL OneTest™ Server after executing the command.

Note: The following fields are from HCL® Launch:

- **Working Directory**
- **Precondition**
- **Post Processing Script**



- **Use Impersonation**
- **Auth Token Restriction**

You can accept the default values or change the values based on your requirements. For more information about these fields, see the related links.

c. Click **OK** to save the properties for the test.

9. Click **Save** in the design area.

Results

You have configured the process for the component in HCL® Launch.

What to do next

You must create a resource in HCL® Launch. See Creating a resource in HCL Launch on page 442.

Related information

Creating a command-line config file on page 1400

Viewing test logs on page 1493

Generating offline user tokens

HCL Launch Documentation

Process step preconditions

Post-processing scripts

User impersonation for process steps

Restricting authentication tokens

Creating a resource in HCL® Launch

You must create a resource to associate agents with components that you created in HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Resources**, and then click **Create Top-Level Group**.
   The **Create Resource** dialog is displayed.
3. Enter a name for the resource in the **Name** field.
4. Click **Save**.

Results

You have created the resource in HCL® Launch.

What to do next

You must configure the resource. See Configuring the resource on page 443.

Related information

HCL Launch Documentation

Configuring the resource

You must configure the resource to add an agent and associate the agent with the component.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 429.
  - Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 442

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Resources**.
   Result
   A list of resources that are available in HCL® Launch is displayed.
3. Perform the following steps to add an agent to the resource:
   a. Click the resource from the list for which you want to add an agent.
   b. Click the **Actions** icon from the last column, and then click **Add Agent**.
   c. Select the agent from the drop-down list.

      Note: The **Name** field is auto populated with the name of the agent.

   d. Click **Save**.
      Result
      The selected agent is added to the resource and you can view the status of the agent in the **Status** column.
4. Perform the following steps to add a component to the agent:

a. Click the agent from the list for which you want to add a component.

b. Click the **Actions** icon from the last column, and then click **Add Component**.

c. Select the component from the drop-down list.

Note: The **Name** field is auto populated with the name of the component.

d. Click **Save**.

Result

The selected component is added to the agent.

Results

You have configured the resource in HCL® Launch.

What to do next

You must create an application. See Creating an application in HCL Launch on page 444.

Related information

HCL Launch Documentation

Creating an application in HCL® Launch

You must create an application to fetch all the components together that you want to deploy.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
3. Click **Create Applications**, and then **New Applications**.
4. Enter a name for the application in the **Name** field.
   The **Environments** page for the application that you created is displayed.

Results

You have created the application in HCL® Launch.

What to do next

You must configure the application. See Configuring the application on page 445.

Related information

HCL Launch Documentation

Configuring the application

You must configure the application to associate resources with environments and define processes to run tests.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 429.
  - Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 429.
  - Configure the process for the component in HCL® Launch. See Configuring the process on page 430 .
  - Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 442
  - Configured the resource in HCL® Launch. See Configuring the resource on page 443.
  - Created an application in HCL® Launch. See Creating an application in HCL Launch on page 444.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
   Result
   A list of applications that are available in HCL® Launch is displayed.
3. Click the application that you want to configure from the **Name** column.
   The **Environments** page for the selected application is displayed.
4. Perform the following steps to create an environment for the application that you selected:
   a. Click **Create Environment**.
   b. Enter a name for the environment in the **Name** field.
   c. Click **Save**.
5. Perform the following steps to configure resources to the environment:
   a. Click the environment that you created.
   b. Click **Add Base Resources**.
      Result
      A list of resources that are available in HCL® Launch is displayed.
   c. Select the checkbox to add resources to the environment.
   d. Click **Save**.

Result

You can view the corresponding agent and the component that you added for the resource by using the **Expand** icon.

6. Perform the following steps to add the component to the application:
    a. Click **Applications**, and then select your application from the list.
    b. Click the **Components** tab, and then **Add Components**.
    c. Select the checkbox from the drop-down list to add components to the application.
    d. Click **Save**.
7. Perform the following steps to create a process for the application:

    a. Click the **Processes** tab, and then **Create Process**.

    b. Enter a name for the process in the **Name** field.

    c. Click **Save**.
    Result
    The **Design** tab for the process that you created is displayed.

8. Drag the component process listed under the **Component Process Steps** option from the left navigation pane and drop it into the design area.
9. Select the component process from the drop-down list in the **Operational (No Version Needed) Process** field.
10. Click **Save**.
11. Click the **Edit** icon, and then change the name of the properties.
12. Click **OK**, and then click **Save**.

Results

You have configured the application to run tests from HCL® Launch.

What to do next

You can run tests from HCL® Launch. See Running tests from HCL Launch on page 446.

Related information

HCL Launch Documentation

Running tests from HCL® Launch

You can run the Web UI tests for your application under test from HCL® Launch as a **Process**.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:

- Been granted access to HCL® Launch.
- Created a component in HCL® Launch. See Creating a component in HCL Launch on page 429.
- Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 429.
- Configure the process for the component in HCL® Launch. See Configuring the process on page 430 .
- Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 442
- Configured the resource in HCL® Launch. See Configuring the resource on page 443.
- Created an application in HCL® Launch. See Creating an application in HCL Launch on page 444.
- Started the HCL® Launch agent. For more information, refer to the HCL Launch documentation.
- Closed Rational® Functional TesterHCL OneTest™ UI.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
   Result
   A list of applications that are available in HCL® Launch is displayed.
3. Click the application that you configured for the test run.
4. Click the **Request Process** icon.
   Result
   The **Create Deployment** page is displayed.
5. Select the process that you created for the application which contains the test assets from the **Process** drop-down list.
6. Click **Next**, and then select the specific version of component, if any.

   Note: If you have multiple versions for your component, then only you can choose the required version by clicking the **Add** option.

7. Click **Next**, and then verify the properties for the test run.
8. Click **Next**, and then verify that the status of the **Run Now** option is enabled.
   You can disable the **Run Now** option, and then use the **Date**, **Time**, and **Recurrence Pattern** fields to run tests as **Process** at the scheduled time.
9. Click **Next**, and then verify the details about the **Process**.
10. Click **Submit Deployment** to initiate the test run.

Results

You have run the Web UI test as **Process** from HCL® Launch.

What to do next

You can view the status of the process as follows:

- **Success**: When the test run is successful
- **Failed**: When the test run fails

You can view the details of the test run in the **Log** tab. The **Reports information** section on the Log file displays the names of the report along with its corresponding URLs if you publish test results to IBM® Rational® Test Automation ServerHCL OneTest™ Server.

Testing with UrbanCode Deploy

With the IBM® Rational® Functional TesterHCL OneTest™ UI plugin for UrbanCode™ Deploy, you can automate the execution of tests and compound tests.

Before you begin

You must have completed the following tasks:

- Installed Rational® Functional TesterHCL OneTest™ UI and UrbanCode™ Deploy agent on the computer where the tests will be run.
- Downloaded the Rational® Functional TesterHCL OneTest™ UI UCD plugin `RFT-UCD-UITest-10.1HOT-UI-UCD-UITest-10.1` from the IBM WebSphere, Liberty & DevOps Community portal HCL® License & Delivery portal.
- Installed UrbanCode™ Deploy server and deploy the plugin on the server. For information, refer to the UrbanCode Deploy documentation.
- Installed UrbanCode™ Deploy Agent and connect it to UrbanCode™ Deploy server. For more information refer to the UrbanCode Deploy documentation.
- Ensured that Rational® Functional TesterHCL OneTest™ UI is not running.
- Ensured that all the devices are set to the passive mode and the playback-ready apk files are installed to initiate the mobile test runs.

Note:

- To run tests on Mac OS, you must add an environment variable that points to the installation directory of the product, for example, `export TEST_WORKBENCH_HOME=/opt/IBM/SDPexport TEST_WORKBENCH_HOME=/opt/HCL/HCLOneTest`. For Windows™, this environment variable is already defined.
- If you installed the product on Windows using the stand-alone installer (not using Installation Manager), you must specify the environment variable that points to the installation directory of the product. For example, `export TEST_WORKBENCH_HOME=C:\Program Files\HCL\HCLOneTest`

About this task

As a tester, you might have a large number of regression tests to run on the latest build of the product. Instead of manually running the tests on every new build, you can install the latest build on the UrbanCode™ Deploy Agent computer and let UrbanCode™ Deploy launch the tests for you.

After deploying the UrbanCode™ Deploy plugin on the server, create the component and its processes, applications and its processes, the environments, and the resources. For information about how to create these different pieces, see UrbanCode™ Deploy documentation.

To create the workflow:

1. To create a workflow for the newly created component process, click the component process. The Tools view displays the available plugin steps.
2. From **Rational Functional Tester - UI Test > HCL OneTest UI - UI Test > Run an IBM Rational Functional Tester - UI test > Run an HCL OneTest UI - UI test** to the design space.
3. Specify the properties for the step. For information about the properties, see Properties of Steps on page 449.
4. After you configure the step properties, save the step by clicking **Save**.

Exemple

You can run a process with multiple tests.

If you have installed multiple products on one target machine, either in the same or different package groups, you can use a single process that includes steps from different products. You must install the plugins of all of the products that you want to use.

Properties of Steps

You must set the properties of steps to run the tests.

Input properties for the test run step

You must provide the values for the following required fields:

- **Workspace**
- **Project**
- **Test Suite Name**

If you include these required parameters in a configuration file and use the **Config File** field to specify the complete file path, then these values are not required.

Field	Description
Required	
Name	Enter the name of the step.

Field	Description
Workspace	Enter the complete path to the Eclipse workspace.
Project	Enter the path, including the file name of the project relative to workspace.
Test Suite Name	Enter the path, including the file name, of the test to run relative to the project. A test can be a Web UI test, compound test, or Accelerated Functional Test. To run multiple tests sequentially, specify test names separated by comma.
Optional	
IMShared Location	Enter the complete path to the location of the IBMIMSharedHCLIMShared folder. For example, `C:\Program Files\IBM\IBMIMSharedC:\Program Files\HCL\HCLIMShared`
Import	Enter the complete path to the project folder that is cloned from a source control system or remote repository. When you use this option, you can specifiy a name for the new workspace to be created to run the imported test assets in the Workspace field.
Var File	Enter the complete path to the XML file that contains the variable name and value pairs.
Config File	Enter the complete path to a file that contains the parameters for a test run.
Results File	Enter the name of the results file. The default result file is the test name with a time stamp appended.
Overwrite Results file	Determines whether a results file with the same name is overwritten. The default value is true which indicates that the results files can be overwritten.
Number of Virtual Users	For a schedule, the default value is the number of users specified in the schedule editor. For a test, the default value is one user. Overrides the default number of users, if required.
VM Args	Java™ virtual machine arguments to pass in.
Protocol Input	Use this argument to run a Web UI test in parallel on different browsers. `-protocolinput "all.available.targets.in.parallel=all"`

Field	Description

	`-protocolinput "all.available.targets.in.parallel=chrome,ff,ie"`
	
	Note: If you use the `-protocolinput` argument, you must not use the equivalent -vmargs arguments:
	`-vmargs "-Dall.available.targets.in.parallel=all" -vmargs "-Dall.available.targets.in.parallel=browser1,browser2,browser3"`

Dataset Override	For a test, the default value is the dataset specified in the test editor. **Note:** You must use the `Dataset Override` option to replace the dataset values during a test run. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset. For example, `/project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv`. You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a `semicolon (;)`. For example, `/project_name1/ds_path/ds_filename.csv:/project_name1/ds_path/new_ds_filename.csv; /project_name2/ds_path/ds_filename.csv:/project_name2/ds_path/new_ds_filename.csv`
Exported HTTP Test log File	Enter the complete path to a file in which to store the exported HTTP test log.

Field	Description
Exported Statistical Report Data File	Enter the complete path to the directory in which to store exported statistical report data.
Custom Report Format Files	A comma-separated list of absolute paths to custom report format files (.view files) to use when exporting statistical report data with the -exportstats option.
User Comments	Enter the text within double quotation mark to display it in the **User Comments** row of the report.  **Note:** • When you run tests by using the double quotation marks ("") for the **User Comments** field, then the **User Comments** row of a report does not contain double quotation marks. • To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field.
Export Report	Specify the values to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. Enter the following details in the text field: *type=<report type>;format=<file type>;folder=<destination folder path>;filename=<name of the exported file>* For example, type=unified;format=pdf,xml,html;folder=E:\temp\logs;filename=UCDExecutionReport When you choose to export the report to all the supported formats, the report in different file formats use the same file name that is specified in the field.
Exportstatsformat	Specify a format for the report that you want to export. You must use the **Exported Statistical Report Data File** parameter along with the **Export stats format** parameter. You must use at least one of the following formats:

Field	Description
	• *simple.csv* • *full.csv* • *simple.json* • *full.json* • *csv* • *json* For example, **Exported Statistical Report Data File** = *<local_dir_path>* **Export stats format** = *json* You can add multiple formats for the report separated by a comma (,). If you want to export both the simple and full reports in a *json* or *csv* format, you can specify *json* or *csv* as the format in the field. The reports are saved to the location specified in the **Exported Statistical Report Data File** field.
Labels	Enter the labels to be added to the test results when the test run is complete. You can add multiple labels to a test result separated by a comma. For example, *"label1, label2"* When you run test assets then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. If you have set **Publish result after execution** as **Always** or **Prompt** in the Rational® Functional TesterHCL OneTest™ UI preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server**) and use the **Labels** option, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset. **Note:**

Field	Description
	 • When you run tests by using the double quotation marks ("") for the **Labels** field, then the labels in the test result do not include the double quotation marks. • To work around this problem, you must create a command-line config file, and then run the test by using the **Config File** field. • When you use the **Config File** field to run tests, then labels provided in the configuration file take precedence over the labels provided in the **Labels** field.
Exportstatshtml	Enter the complete path to a directory to export web analytic results. The results are exported to the specified directory. You can then analyze the results on a web browser without using the test workbench.
Publish	Specify the server URL and project name to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You must provide the URL and offline user token of the server in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server** of Rational® Functional TesterHCL OneTest™ UI before you use the **publish** parameter in the test script. Use the following arguments with the **Publish** parameter: • To specify the project name, use any of the following formats: ◦ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ◦ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias*

Field	Description

	You must consider the following points while providing the project name:
	◦ If the project name is not specified, then the value of the **Project** parameter is used.
	◦ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias=**=*name_of_the_teamspace_alias* options.
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = https://localhost:5443#project.name=test&teamspace.name=ts1`
	Where:
	▪ https://localhost:5443 is the URL of the server.
	▪ test is the name of the project.
	▪ ts1 is the name of the team space.
	◦ If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`.
	For example, if the name of the team space is *Initial Team Space*, then you

Field	Description

	must provide it as *Intial%20Team %20Space*.
	
	Where, %20 is the hexadecimal value of the space character.
	
	• To avoid publishing of reports, use **no**.
	
	You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them.
	
	For example:
	
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish = no`
	
	If you do not use the **Config File** parameter to run the tests, then the values provided in the test script always take precedence over the **Results** options set in the product preferences (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).
	
	The **Reports information** section on the **Output** window displays the names of the report along with its corresponding URLs in the following conditions:
	
	• When you configured the URL of Rational® Test Automation ServerHCL OneTest™ Server in preferences of Rational® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server**.
	
	• When you set **Publish result after execution** as **Always** or **Prompt** in the preferences of Ratio-

Field	Description
	nal® Functional TesterHCL OneTest™ UI (**Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results**).

Publish_for	Specify the option to publish test results to Rational® Test Automation ServerHCL OneTest™ Server based on the completion status of the tests. You must use the **Publish_for** parameter along with the **Publish** parameter.
	
	The following are the available options that you can use for the **Publish_for** parameter:
	
	• **ALL** - This is the default option. You can use this option to publish test results for any text execution verdict.
	• **PASS** - You can use this option to publish test results for the tests that have passed.
	• **FAIL** - You can use this option to publish test results for the tests that have failed.
	• **ERROR** - You can use this option to publish test results for the tests that included errors.
	• **INCONCLUSIVE** - You can use this option to publish test results for the inconclusive tests.
	
	You can add multiple parameters separated by a comma.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish =`
	` https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publish_for = FAIL,ERROR`

Publishreports	Specify the option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. The values that you can use with **Publishreports** are as follows:

Field	Description

	• FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455.
	• STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469.
	• TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491.
	
	You must use the **Publishreports** parameter along with the **Publish** parameter.
	
	For example:
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish =`
	` https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publishreports = STATS, TESTLOG`
	
	The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI.
	
	You can prefix the value with *"!"* to publish the reports except for the specified one in the test script.
	
	For example,
	`Workspace = C:/Users/IBM/rationalsdp/workspace1`
	`Project = proj1`
	`Test Suite Name = Tests/testHttp.testsuite`
	`Publish =`
	` https://localhost:5443#project.name=test&teamspace.name=ts1`
	`Publishreports = !TESTLOG`
	
	All the reports except the TESTLOG report is published to Rational® Test Automation ServerHCL OneTest™ Server after executing the command.

Related information

Creating an Accelerated Functional Test asset on page 746

Integrations in Functional Test perspective

In this section, you will learn about the supported integrations for the Functional Test perspective.

Integration plugin compatibility matrix

You can find information about the versions of the integration plugins that are required to integrate Jenkins, Ant, HCL® Launch, and UrbanCode™ Deploy with Rational® Functional TesterHCL OneTest™ UI.

The following table lists the versions of the integration plugins for the Functional Test persepctive.

Note: You must download the required version of the integration plugin from the IBM WebSphere, Liberty & DevOps Community portalHCL® License & Delivery portal based on the existing version of Rational® Functional TesterHCL OneTest™ UI. You can then integrate Jenkins, Ant, HCL® Launch, and UrbanCode™ Deploy with Rational® Functional TesterHCL OneTest™ UI.

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version
10.1.0	RFT-Ant-2.0	RFT-Jenkins-5.0	RFT-UCD-5.0
10.1.1	RFT-Ant-2.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.1.2	RFT-Ant-3.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.1.3	RFT-Ant-3.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.2.0	RFT-Ant-3.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.2.1	RFT-Ant-3.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.2.2	RFT-Ant-3.0	RFT-Jenkins-6.0	RFT-UCD-5.0
10.2.3	RFT-Ant-3.1	RFT-Jenkins-7.1	RFT-UCD-5.2

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version
10.5.0	RFT-Ant-FunctionalTest-3.1	RFT-Jenkins-Functional-Test-7.2	RFT-UCD-Functional-Test-6.0
10.5.1	RFT-Ant-FunctionalTest-3.1	RFT-Jenkins-Functional-Test-7.3	RFT-UCD-Functional-Test-6.1
10.5.2	RFT-Ant-FunctionalTest-3.1	RFT-Jenkins-Functional-Test-7.4	RFT-UCD-Functional-Test-6.1
10.5.3	RFT-Ant-FunctionalTest-3.1	RFT-Jenkins-Functional-Test-7.5	RFT-UCD-Functional-Test-6.1

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version	HCL® Launch plugin version
10.1.0	HOT-UI-Ant-3.0	HOT-UI-Jenkins-5.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.1.1	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.1.2	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.1.3	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.2.0	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.2.1	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.2.2	HOT-UI-Ant-3.0	HOT-UI-Jenkins-6.0	HOT-UI-UCD-5.0	HOT-UI-LAUNCH-5.0
10.2.3	HOT-UI-Ant-3.1	HOT-UI-Jenkins-7.1	HOT-UI-UCD-5.2	HOT-UI-LAUNCH-5.2
10.5.0	HOT-UI-Ant-FunctionalTest-3.1	HOT-UI-Jenkins-FunctionalTest-7.2	HOT-UI-UCD-FunctionalTest-6.0	HOT-UI-LAUNCH-FunctionalTest-6.0

Rational® Functional TesterHCL OneTest™ UI version	Ant plugin version	Jenkins plugin version	UrbanCode™ Deploy plugin version	HCL® Launch plugin version
10.5.1	HOT-UI-Ant-FunctionalTest-3.1	HOT-UI-Jenkins-FunctionalTest-7.3	HOT-UI-UCD-FunctionalTest-6.1	HOT-UI-LAUNCH-FunctionalTest-6.1
10.5.2	HOT-UI-Ant-FunctionalTest-3.1	HOT-UI-Jenkins-FunctionalTest-7.4	HOT-UI-UCD-FunctionalTest-6.1	HOT-UI-LAUNCH-FunctionalTest-6.1
10.5.3	HOT-UI-Ant-FunctionalTest-3.1	HOT-UI-Jenkins-FunctionalTest-7.5	HOT-UI-UCD-FunctionalTest-6.1	HOT-UI-LAUNCH-FunctionalTest-6.1

Testing with Ant

You can use `ant` to run functional tests from the command line. Starting with version 2.0 of the `ant` plugin, you can run multiple tests simultaneously.

Before you begin

- Install Installation Manager.
- Install Rational® Functional TesterHCL OneTest™ UI.
- Verify that you have a functional test residing within an Eclipse workspace on the computer where Rational® Functional TesterHCL OneTest™ UI is installed.
- Be sure `ant` is installed and added to the PATH environment variable.
- Download the `ant` plugin for Rational® Functional TesterHCL OneTest™ UI from Rational® Functional TesterHCL OneTest™ UI PluginsHCL® License & Delivery portal and install Ant on to the computer where Rational® Functional TesterHCL OneTest™ UI is installed.

1. Extract the following files from the downloaded plugin:
   - `RFT-Ant-FunctionalTest-3.1.jarHOT-UI-Ant-FunctionalTest-3.1.jar`
   - `RFT-Ant-FunctionalTest.xmlHOT-UI-Ant-FunctionalTest.xml`
   - `README.txt`
2. Open the `RFT-Ant-FunctionalTest.xmlHOT-UI-Ant-FunctionalTest.xml` file and provide parameter values, as shown in the following example:

   ```
   <ft name="test1" projectDir="C:\workspace\Project" scriptName="Script1" />
   ```

   To run multiple tests, add additional `<ft>` tasks and provide details for each test.

   The following table explains each parameter.

Parameter	Description
-name	Required. Name of the test.

Parameter	Description
- projectDir	Required. The fully qualified path to the Rational® Functional TesterHCL OneTest™ UI project directory. Use '\\' or '/' as the file separator.
- scriptName	Required. The name of the script to be run.
- logFormat	Optional. The format of the logs that are created when the script is run. The options are: Default, XML, HTML, text, and TPTP.
- iterationCount	Optional. The number of dataset iterations to be run.
- userArgs	Optional. Additional playback arguments, if any.

3. Open a command prompt as an administrator or a root user. This applies even when you have logged into a test machine with administrator privileges.
4. Navigate to the download directory and verify that it includes the `RFT-Ant-FunctionalTest.xmlHOT-UI-Ant-FunctionalTest.xml` file.
5. Type `ant -f RFT-Ant-FunctionalTest.xmlHOT-UI-Ant-FunctionalTest.xml` to start test execution.

   When `ant` execution completes, a test result is displayed. Ant execution output is logged into the `logfile.txt` file, and a test log is created in a temp directory under `RFT-Ant-FunctionalTest-3.1 HOT-UI-Ant-FunctionalTest-3.1` from where the `ant` command is executed.

Testing with Azure DevOps for functional tests

When you use Azure DevOps for continuous integration and continuous deployment of your application, you can create tests for your application in Rational® Functional TesterHCL OneTest™ UI and run those tests in Azure DevOps pipelines. You can integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension that is available in the **Visual Studio Marketplace** portal.

Prerequisites

Before you integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI, you must have completed certain tasks. See Prerequisites for Azure DevOps Integration on page 462.

Overview

You can use the *IBM Rational Test WorkbenchHCL OneTest Studio* extension that enables you to select any type of test created in Rational® Functional TesterHCL OneTest™ UI that you can add to your task for the job in the Azure DevOps pipelines.

Prerequisites for Azure DevOps integration with Rational® Functional TesterHCL OneTest™ UI

Before you integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension, you must have completed certain tasks.

- You must have installed Rational® Functional TesterHCL OneTest™ UI on a computer running Windows™ or Linux®.
- You must have created an organization and a project in Azure DevOps for running jobs in Azure DevOps pipelines. For more information refer to Creating an organization.

You can now follow the tasks listed in the task flow table to integrate Azure DevOps with Rational® Functional TesterHCL OneTest™ UI. See Task flow for integrating Azure DevOps on page 463.

Task flow for integrating Azure DevOps with Rational® Functional TesterHCL OneTest™ UI

The table shows the task flow for integrating Azure DevOps with Rational® Functional TesterHCL OneTest™ UI by using the *IBM Rational Test WorkbenchHCL OneTest Studio* extension. You must perform these tasks in sequence as listed in the following table. The table also provides you the links to the information about the tasks.

	Tasks	More information
1	Create any or all of the following types of tests in Rational® Functional TesterHCL OneTest™ UI to test your application: • Accelerated Functional Testing (AFT) Suites • Web UI tests • Compound tests • Traditional functional tests	Testing in the Functional Test perspective on page 773
2	Access the **Visual Studio Marketplace** portal and search for the latest version of the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.	Visual Studio Marketplace
3	Install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.	Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension on page 463
4	Run tests in an Azure DevOps pipeline.	Running tests in an Azure DevOps Pipeline on page 465

Related information

Testing with Azure DevOps for functional tests on page 462

Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension

You must install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization before you use the extension to run tests for your application in an Azure DevOps pipeline. The *IBM Rational Test*

WorkbenchHCL OneTest Studio extension supports running of tests created in Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have access to the **Visual Studio Marketplace** portal.

About this task

After you install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension from the **Visual Studio Marketplace** portal in your Azure DevOps organization, you can use the extension to run tests for your application in an Azure DevOps pipeline.

1. Log in to the **Visual Studio Marketplace** portal, if you are not already logged in.
2. Click the **Azure DevOps** tab.
3. Search for the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
4. Click the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
5. Click **Get it free**.
   Result

   The **Visual Studio Marketplace** portal for the *IBM Rational Test WorkbenchHCL OneTest Studio* extension is displayed.
6. Select the organization where you want to run your test from the **Select an Azure DevOps Organization** list.
7. Click **Install**.
   Result

   The installation is completed.
8. Click **Proceed to organization**.
   Result

   The **Organization** page in Azure DevOps is displayed.
9. Click **Organization settings > Extensions**.
   Result

   The *IBM Rational Test WorkbenchHCL OneTest Studio* extension is displayed as an installed extension.

Results

You installed the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization.

What to do next

You can add tests that you created in Rational® Functional TesterHCL OneTest™ UI to your task, and then run the tests in an Azure DevOps pipeline. See Running functional tests in an Azure DevOps Pipeline on page 465.

Running functional tests in an Azure DevOps Pipeline

After you create the tests in IBM® Rational® Functional TesterHCL OneTest™ UI for the application that you are testing, and after you install the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your organization, you can run the tests in Azure DevOps pipelines.

Before you begin

You must have completed the following tasks:

- Installed the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your organization. See Installing the *IBM Rational Test WorkbenchHCL OneTest Studio* extension on page 463.
- Installed an agent in your pipeline. See Azure Pipelines agents.

About this task

After you add the *IBM Rational Test WorkbenchHCL OneTest Studio* extension in your Azure DevOps organization, you can use an existing pipeline or create a new one to add Rational® Functional TesterHCL OneTest™ UI test tasks. You can install an agent or use the one that you installed in your default agent pool. You can add the Rational® Functional TesterHCL OneTest™ UI tests to your task for the agent job, configure the task, and then run the task in the Azure DevOps pipeline.

If you have created test cases under test plans in your Azure DevOps project, you can provide the details of the Azure DevOps URL, test plan, test case, and your personal access token (PAT) while you configure the test job in a pipeline so that you can view the results of the test run on your Test Plan dashboard.

1. Open your **Organization** page in Azure DevOps and perform the following steps:
    a. Click the project you want to use.
    b. Initialize the repository by performing the following steps:
        i. Click **Repos** from the left pane.
        ii. Click **Initialize** from the **Initialize with a README or gitignore** section.

        Note: Select the **Add a README** check box if it is not selected.

    c. Click **Pipelines** from the left pane.
    d. Click **Create Pipeline**.
    e. Click **Use the classic editor** to create a pipeline without YAML.
    f. Verify the project, repository, and branch for manual and scheduled builds, and then click **Continue**.
    g. Click **Empty job**.
2. Select **Pipeline** and complete the following steps:

a. Change the name for the build pipeline if required.

b. Select the **Agent pool** for your build pipeline.

You can use the agent from the default agent pool or use the one you have installed.

c. Select the **Agent Specification** for the agent if required.

3. Add a task to the agent job by completing the following steps:

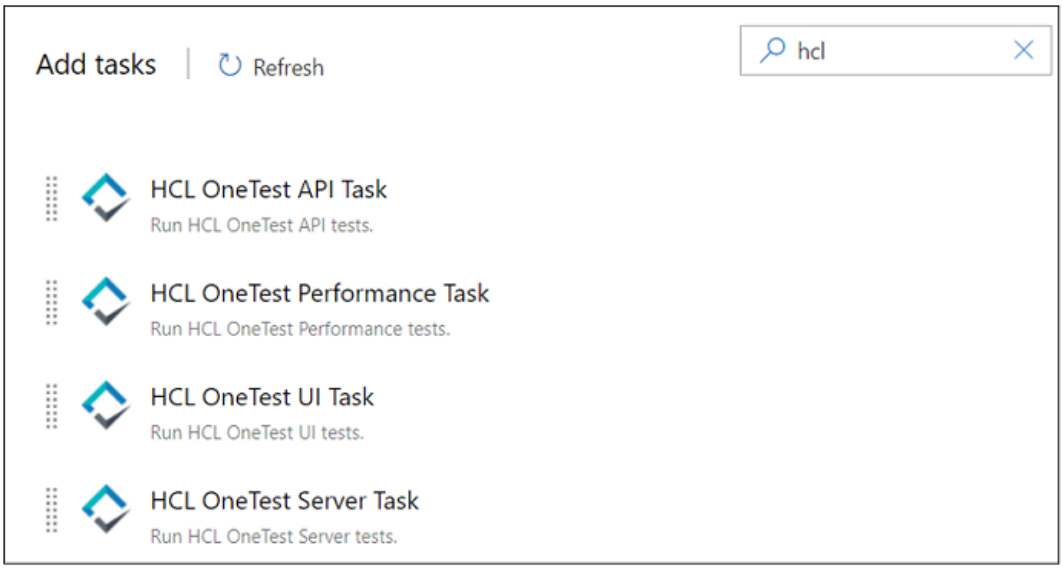
a. Click the **Add Task** icon for the agent job.
Result

The **Add tasks** pane is displayed.

b. Search for the IBMHCL tasks defined in the *IBM Rational Test WorkbenchHCL OneTest Studio* extension.
Result

The tasks that you can select are displayed.





Depending on the type of test that you have created in Rational® Functional TesterHCL OneTest™ UI, you can select the type of task. You must use the following table to identify the task you must select:

Type of test	Task to select
Functional test scripts	**Rational Functional Tester TaskHCL OneTest UI Task**

c. Select the **IBM Rational Functional Tester TaskHCL OneTest UI Task** option, and then click **Add** to add the task to the agent job.
Result

The selected task is added to the agent job and it is displayed with a warning that some settings require attention. You must configure the settings mentioned in Step 4 on page 467.

You can also remove the tasks that are not required in your job. Select the tasks in the list that you want to remove. You can then right-click the tasks, and click **Remove selected task(s)** to remove them.

4. Configure the settings by performing the following steps:

a. Select the task version from the list if required.

b. Follow the action for the functional tester task by referring to the following table:

Note: All mandatory fields are marked with an asterisk (*) in the UI.

Field	Description	Action
Display name	Displays the name of the selected task.	Enter the name of the task.
Testcase Type	The type of test to execute.	Select **Traditional (Java)** from the **Test-case Type** list.
Product Path	The fully qualified path to the Rational® Functional TesterHCL OneTest™ UI. This path must exist on the agent computer.	Enter the complete path of Rational® Functional TesterHCL OneTest™ UI.
Project Name	The name of the project containing the test.	Enter the name of the project containing the test.
Project Di-rectory	The path where the project containing the test is located on your computer.	Enter the complete path to the location of the project containing the test. For exam-ple, `D:\Projects\`

Field	Description	Action
Test Suite Name	The name of the functional test script that you execute without the file extension.	Enter the name of the script that you want to run. For example, `script1`.
Export Report	The option to export the unified report to other file formats such as PDF, XML, and HTML.	Enter the following details in the text field: type=<report type>;format=<file type>;folder=<destination folder path>;filename=<name of the exported file> For example, to export the report to only the pdf format, you can use type=unified;format=pdf;folder=Expreport1;filename=testreport . If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use type=unified;format=pdf,xml,html;folder=Expreport1;filename=testreport.
Log Format	The format of the script logs.	Select the format of the script log from the list. **Note:** By default, `Default` is selected.
Iteration Count	The number of datasets iterations to be run.	Enter the number that you want for the dataset iterations.
User Arguments	You can specify additional arguments that you want to run in the test.	Specify any arguments for the test run. For example, you can specify the playback arguments, if applicable in your test.

Field	Description	Action
Azure DevOps Project URL	The URL of the test project in the organization on the Azure DevOps server. **Note:** You must enter the details for this option if you have created test cases under Test plans in your Azure project and want to view the test results on the Test Plans Dashboard.	Enter the URL of the test project in the organization on the Azure DevOps server in the following format: https://<host>/<orgname>/<projectName> You must use this option with the following options: ▪ Azure DevOps PAT ▪ Test Plan Name ▪ Test Case Name
Azure DevOps PAT	The personal access token (PAT) of the Azure server where the test project that contains the test plan is hosted. You can also enter the variable for the encrypted PAT that you create in Azure DevOps. **Note:** The token must have read and write access.	Enter your personal access token. You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Test Plan Name ▪ Test Case Name
Test Plan Name	The name of the test plan in the Azure server.	Enter the name of the test plan. You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Azure DevOps PAT ▪ Test Case Name
Test Case Name	The name of the test case in the Azure server.	Enter the name of the test case. You must use this option with the following options: ▪ Azure DevOps Project URL ▪ Azure DevOps PAT ▪ Test Plan Name

c. Expand **Control Options** and configure the settings for your task if required.

d. Expand **Output Variables** and configure the settings for your task if required.

5. Select the following options:

a. Click **Save** to save the configured settings for the task.



Note: The task is not queued for a run.

You can save the task to a build pipeline and opt to run the build at a later time.

b. Click **Save & queue** to save the configurations and queue the run in the pipeline.
Result

The **Run pipeline** dialog box is displayed.

6. Complete the following steps:

a. Enter a comment for the test in the **Save comment** field.

b. Select the agent that you configured for the test from the **Agent pool** list.

c. Select the agent specification from the **Agent Specification** list for the agent if required.

d. Select the branch from the **Branch/tag** list.

e. Add the variables and demands for the task run from the **Advanced Options** pane if required.

f. Select the **Enable system diagnostics** check box for a detailed log view.

g. Click **Save and run**.
Result

The **pipeline summary** page displays the progress of the job run.

Results

You have run the tests for the application you are testing, in the Azure DevOps pipeline.

What to do next

- You can open the job to view the task logs from the **pipeline summary** page.
- You can click **Test Plans** to view the test result if you specified the values for **Azure DevOps Project URL**, **Azure DevOps PAT**, **Test Plan Name**, and **Test Case Name**.
- You must click the task to open the **Task** page to view the test results.
- You can access the report URLs to view the test execution information at any point of time. The report URLs are the IBM® Rational® Test Automation ServerHCL OneTest™ Server URLs where the reports are stored.

  In Rational® Functional TesterHCL OneTest™ UI, if the Rational® Test Automation ServerHCL OneTest™ Server URL is configured in **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest**

Server and **Publish result after execution** is set as **Always** in **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server > Results**, then the logs in the **Task** page also displays the names of the published report along with its corresponding URLs.

Related information

Configuration of test runs from the command line on page 1381

Testing with Cucumber

Starting with Rational® Functional TesterHCL OneTest™ UI 9.2, you can take advantage of Cucumber integration to create, annotate, and run functional test scripts that can be easily understood by all members of the team.

About Cucumber

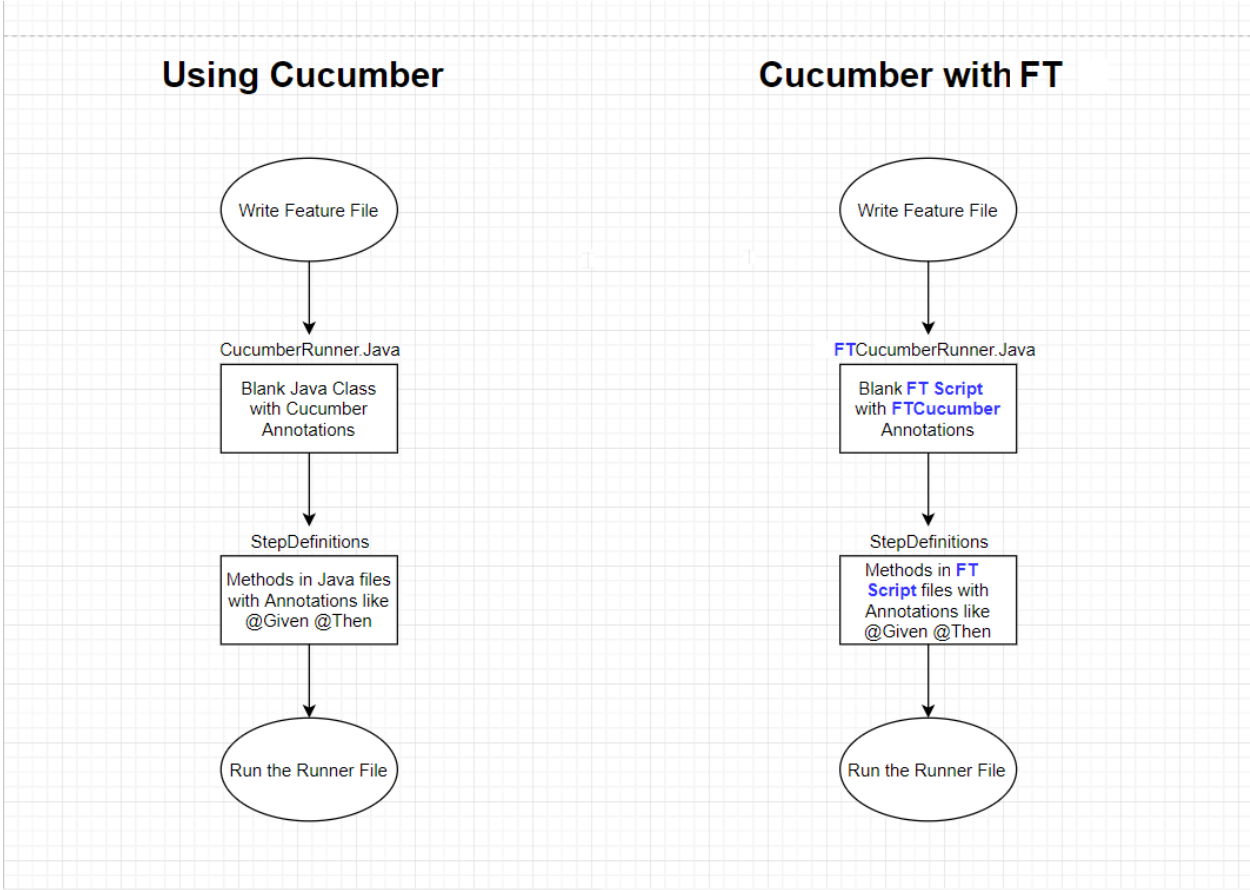
Cucumber is a software testing tool that fosters better communication among domain experts, business analysts, testers, and developers by providing everyone a clear view of the testing effort. Cucumber is based on the principles of behavior-driven development (BDD). See Behavior Driven Development for more information.

A functional test script for the Eclipse version of Rational® Functional TesterHCL OneTest™ UI is a collection of Java™ methods. You must be able to read and understand the Java™ code to completely understand the test script. With Cucumber integration, anyone can understand the test script because the flow of the test is described in simple, English statements within a Cucumber feature file.

Note: Rational® Functional TesterHCL OneTest™ UI datasets are not supported in Cucumber feature files. However, Cucumber provides support for its own dataset-like data structures.

The basic steps for using Cucumber with Rational® Functional TesterHCL OneTest™ UI are as follows:

1. Create a *Feature* file, a text file with a `.feature` extension.
2. Create a *Runner* File, an empty functional test script, and annotate the Class in the test script with `@FTCucumberOptions` annotations. Any functional test script that is annotated becomes a runner file.
3. Provide *Step definitions* using Cucumber annotations, such as `@When`, `@Given`, `@Then`, and so on.
4. Run the Runner File, either from Rational® Functional TesterHCL OneTest™ UI or from the command line.



Requirements

Before using the Cucumber integration in Rational® Functional TesterHCL OneTest™ UI, you must set up Cucumber. There are two ways to set up Cucumber:

- By downloading the required Cucumber `jar` files, and copying them to the `customization` folder
- By running a Maven command to set up Cucumber automatically (Starting in 9.2.1)

Manual setup

To set up Cucumber manually, add the following Cucumber-related jar files to the Rational® Functional TesterHCL OneTest™ UI Customization directory, which on Windows™ is located at `C:\ProgramData\HCL\HOTUI\customization` `C:\ProgramData\IBM\RFT\customization`.

- cucumber-core-1.2.5.jar
- cucumber-java-1.2.5.jar
- cucumber-jvm-deps-1.0.5.jar
- gherkin-2.12.2.jar
- cucumber-html-0.2.3.jar

You can download these jar files from the Central Maven repository at https://mvnrepository.com by searching for them one by one. All of the Cucumber-related libraries are grouped under the `info.cukes` group in the Maven repository at https://mvnrepository.com/artifact/info.cukes. The version numbers vary, depending on when you download the files.

Automated setup in 9.2.1 using Maven

To set up Cucumber with Maven:

1. Verify that you have an Internet connection.
2. You must have installed Maven and set up an environment variable that points to the M2_HOME directory. For more details about setting up Maven, see Testing with Maven on page 393.
3. Verify that the pom_dependency_cucumber.xml file is available from your Maven setup.
4. Run the following command in your command prompt window:

   ```
   mvn -f <path to pom_dependency_cucumber.xml file> dependency:copy-dependencies
    -DoutputDirectory=<FT Customization Directory> -Dmdep.stripVersion=true
   ```

   For example:

   ```
   mvn -f "C:\Users\Win10\Desktop\pom_dependency_cucumber.xml" dependency:copy-dependencies
    -DoutputDirectory="C:\ProgramData\IBM\RFT\customization" -Dmdep.stripVersion=true
   ```

   ```
   mvn -f "C:\Users\Win10\Desktop\pom_dependency_cucumber.xml" dependency:copy-dependencies
    -DoutputDirectory="C:\ProgramData\HCL\HOTUI\customization" -Dmdep.stripVersion=true
   ```

   This command downloads the required Cucumber jars into the Rational® Functional TesterHCL OneTest™ UI Customization directory. Depending on the speed of your Internet connection, this command can take several minutes to complete.

Cucumber feature file

The Cucumber feature file is a text file. It provides an abstraction layer that shields the non-technical user from the underlying code. A feature file describes one feature, but can describe multiple scenarios or test cases. It follows a Given-When-Then format, as follows:

- (Given) some context
- (When) some action is carried out
- (Then) a particular set of observable consequences are obtained.

For example:

- Given the login page is displayed for the application.
- When user attempts to log in with incorrect credentials.
- Then user should be re-prompted for credentials.
- Reset password link should be displayed.

The feature files are saved in a folder named `features` inside the functional test project.

A sample feature file that contains four scenarios is as follows:

```
@ClassicsjavaAFeatures
Feature: Test ClassicsJavaA App
  I want to use this template for my feature file

  @smoke
  Scenario: Open and CLose ClassicsJavaA App
  Given I start ClassicsJavaA App
  Then I Close the App

  @smoke
  Scenario: maximize and restore ClassicsJavaA App
  Given I start ClassicsJavaA App
  Then I maximize the App
  Then I click the same button to restore the App
  Then I Close the App

  @smoke
  Scenario: Use all top Options of ClassicsJavaA App
  Given I start ClassicsJavaA App
  Then I use all the top options of the App
  Then I Close the App

  @sanity
  Scenario: Run ClassicJava App to place Order for CD with given Card Details
  Given I start ClassicsJavaA App
  Then I place order for a CD with Card Details
  |composerName | cdTitle                  | custName      | pass | card | month | year|
  |Haydn        |Symphonies Nos. 99 & 101  | Jack Thompson | pas1 | 1234 |   11  | 24  |
  |Haydn        |Symphonies Nos. 94 & 98   | Rick Tumbler  | pas2 | 4545 |   12  | 25  |
  |Bach         |Violin Concertos          | Sonia Evans   | pas3 | 3333 |   08  |18   |
  |Haydn        |Violin Concertos          | Tony Miatta   |pas4  |5555  |   06  |19   |
  Then I Close the App
```

Cucumber Runner file

A Rational® Functional TesterHCL OneTest™ UI Cucumber Runner file is a functional test script that includes `@FTCucumberOptions` annotations. The `@FTCucumberOptions` annotations must be declared outside of the class definition, as shown in the following figure:

```
package Runner;
import resources.Runner.Runner_orderHelper;
/**
 * Description   : Functional Test Script
 * @author sravankumarreddy.k
 */

@FTCucumberOptions(cucumberOptions =
{
//"--glue", "",
//"--plugin", "pretty",
//"--monochrome",
//"--dry-run",
//"--tags", "<tag name>",
//"--strict",
"Features\\orderTest.feature"
        })

public class Runner_order extends Runner_orderHelper
{
    /**
     * Script Name   : <b>Runner_order</b>
     * Generated     : <b>Mar 16, 2018 12:04:14 AM</b>
     * Description   : Functional Test Script
     * Original Host : WinNT Version 6.1  Build 7601 (S)
     *
     * @since  2018/03/16
     * @author sravankumarreddy.k
     */
    public void testMain(Object[] args)
    {
        // TODO Insert code here
    }
}
```

Use the Run button in Rational® Functional TesterHCL OneTest™ UI to run the Runner file, or use the command line. For Cucumber command line options, see https://cucumber.io/docs/cucumber/api/ and look for **List configuration options**

Cucumber Step Definitions

The Step Definition file is a functional test script that contains the code behind each Cucumber annotation in the feature file, that is the @Given, @When, and @Then annotations. An example of a step definitions file for Rational® Functional TesterHCL OneTest™ UI Cucumber integration is as follows:

```
1  package steps;
2  import resources.steps.ClassicsJavaAStepDefsHelper;
24 /**
25  * Description   : Functional Test Script
26  * @author
27  */
28 public class ClassicsJavaAStepDefs extends ClassicsJavaAStepDefsHelper
29 {
30     /**
31      * Script Name   : <b>ClassicsJavaA</b>
32      * Generated     : <b>09-Dec-2017 5:35:22 PM</b>
33      * Description   : Functional Test Script
34      * Original Host : WinNT Version 10.0  Build 15063 ()
35      *
36      * @since  2017/12/09
37      * @author
38      */
39     public void testMain(Object[] args)
40     {
41         // TODO Insert code here
42     }
43
44     @Given("^I start ClassicsJavaA App$")
45     public void i_start_ClassicsJavaA_App() throws Throwable {
46         // Write code here that turns the phrase above into concrete actions
47     }
48
49     @Then("^I Close the App$")
50     public void i_Close_the_App() throws Throwable {
51         // Write code here that turns the phrase above into concrete actions
52     }
53
54     @Then("^I maximize the App$")
55     public void i_maximize_the_App() throws Throwable {
56         // Write code here that turns the phrase above into concrete actions
57     }
58
```

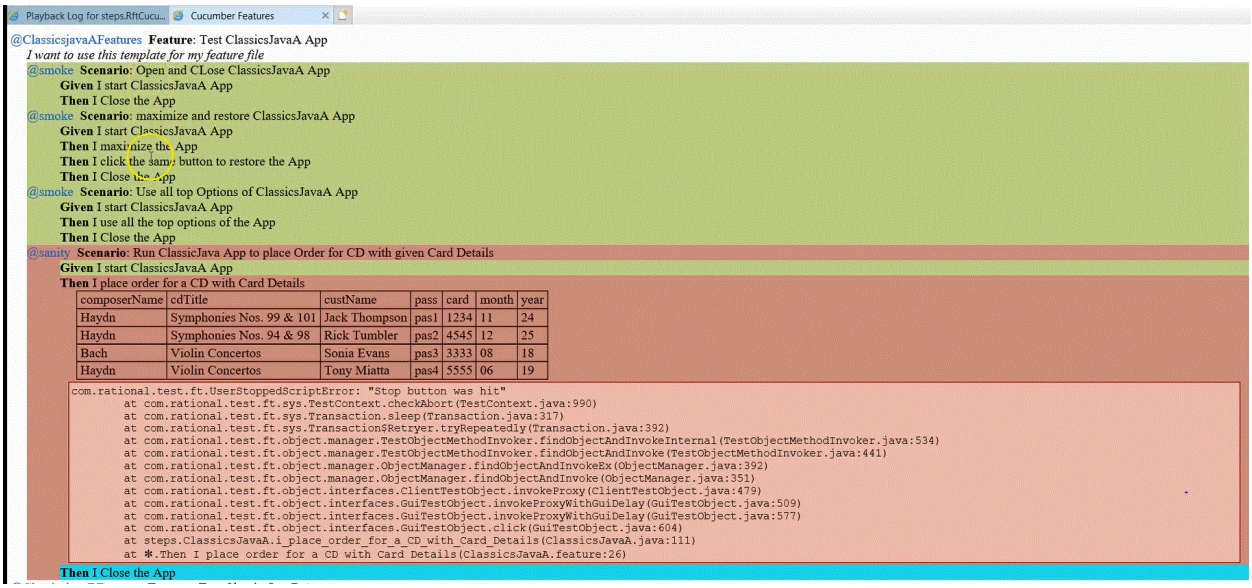
Reports

Two kinds of reports are available:

- Default Cucumber logs
- Extent reports (requires 9.2.1)

Default Cucumber logs

When you run a Rational® Functional TesterHCL OneTest™ UI test script with Cucumber annotations, the log files that are generated contain color-coding to indicate the pass-fail status of each step, as shown in the following example:

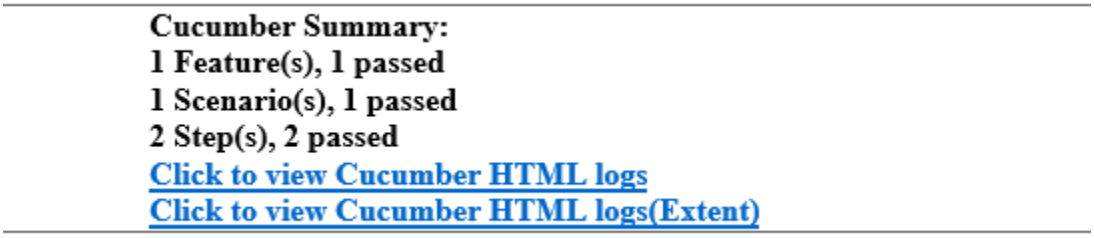


In this example, the steps in red indicate a failure, while the step in blue indicates that the step was not run due to the failure of the previous steps.

The log files also indicate the start point and end point of the feature and each scenario.

Extent reports (requires 9.2.1)

Starting with 9.2.1, you can run the Maven command to enhance Cucumber reporting. For more details about running the Maven command, see pom_dependency_extent.xml. After running this command, you will see an extra option for viewing Extent reports at the end of test execution in the Rational® Functional TesterHCL OneTest™ UI logs, as shown in the following image:



To view the Extent reports, you can run the following command in a command prompt window:

```
mvn -f <path to pom_dependency_extent.xml file> dependency:copy-dependencies -DoutputDirectory=<FT
 Customization Directory> -Dmdep.stripVersion=true
```

For example:

```
mvn -f "C:\Users\Win10\Desktop\pom_dependency_extent.xml" dependency:copy-dependencies
 -DoutputDirectory="C:\ProgramData\IBM\RFT\customization" -Dmdep.stripVersion=true
```

```
mvn -f "C:\Users\Win10\Desktop\pom_dependency_extent.xml" dependency:copy-dependencies
 -DoutputDirectory="C:\ProgramData\HCL\HOTUI\customization" -Dmdep.stripVersion=true
```

Note: The Extent reporting feature is optional. If you run the Extent command, which resolves the dependencies for the Cucumber Extent Report, then the extra link for the Extent report is included in the Rational® Functional TesterHCL OneTest™ UI logs only if you have selected **html** in the **Log type** list, otherwise the link for the Extent report is not included.

Running a feature file with default options

Here is sample code for running a feature file with default options:

```
@FTCucumberOptions( cucumberOptions =
 {
  "cuketest2.feature"  // just providing the folder to the feature files will also work fine.
 })
```

Here are the default options for the other required parameters for the preceding example:

```
--glue '' --plugin pretty --plugin html:CukeLogs --plugin json:CukeLogs/abc.json --monochrome";
```

Required annotations for making a functional test script integrate with Cucumber

Cucumber annotations must be applied to the class. Cucumber options that are passed must be in the same format they would be passed in the Cucumber command line interface. The required annotations include `--glue` and the `.feature` file that needs to be run.

- --glue: Specifies where glue code (step definitions, hooks and plugins) are loaded from. In this case, a blank string ("") as a value to glue tells Rational® Functional TesterHCL OneTest™ UI to search for the step definitions inside all folders in the functional tester project. Duplicate steps lead to exceptions.
- The feature file is in a folder named `Features` inside the functional test project .

```
@code
@FTCucumberOptions(cucumberOptions =
 {
  "--glue", "",
   "Features\\ValidateOrderFeature.feature"
 })
```

Exemple

Providing a folder for the feature files

Run all feature files in the folder named `features` inside the functional test project.

```
@FTCucumberOptions(cucumberOptions =
 {
  "--glue", "",
  "Features"
 })
```

Exemple

Providing more than one feature file

Here is an example with more than one feature file:

```
@FTCucumberOptions(cucumberOptions =
{
 "--glue", "",
 "Features\\PlaceOrder.feature",
 "Features\\ValidateOrder.feature",
})
```

Exemple

Providing two glue (step definition) options

In this case, the step definitions are present in two different folders packages inside the project.

```
@FTCucumberOptions(cucumberOptions =
 {
  "--glue", "com.package1", //Folder inside project- com/package1
  "--glue", "com.package2", //Folder inside project- com/package2
  "Features\\ValidateOrderFeature.feature"
 })
```

Command-line options for running functional test scripts with Cucumber

- Option 1: `-datastore <datastore> -usecucumberoptionscli -playback CucumberRunnerScript`
- Option 2: `-datastore <datastore> -cucumberoptionscli "All Cucumber options as single entry" -playback CucumberRunnerScript`

Option 1 assumes you have already added `FTCucumberOptions` annotations to the functional test script class (a Runner Script). Rational® Functional TesterHCL OneTest™ UI uses these annotations to run as Cucumber.

With Option 2, any options in the Runner Script are ignored, but command-line options are honored.

Option 1 example:`-datastore "C:\work\RFT_WS2\Cuketest3" -usecucumberoptionscli -playback steps.CucumberRunnerScript`

Options 2 example: `-datastore "C:\work\RFT_WS2\Cuketest3" -cucumberoptionscli "--glue 'steps' C:\work\RFT_WS2\Cuketest3\cuketest2.feature" -playback steps.CucumberRunnerScript`

The value for key `cucumberoptionscli` is a single value between double quotes. All of the Cucumber-related command line options go as a single value.

The value for (`--glue`) is either no value between single quotes for all packages ('') or for specific packages, values between single quotes, for example, `'steps'`. For more than one glue entry, specify separate entries as follows: `--glue 'steps' --glue 'moreSteps'`.

Note: Be sure to run these commands from inside the project directory or use the absolute path for the cucumber options feature file and step definition file.

Testing with Micro Focus Application Life Cycle Management

To obtain test result details, you can integrate and run Functional Test scripts in Micro Focus Application Lifecycle Management by using a ready-made template available in the Rational® Functional TesterHCL OneTest™ UI installation directory.

About this task

The Functional Test template is available in the Rational® Functional TesterHCL OneTest™ UI installation directory. You must copy the contents of the template to a new VAPI-XP VBScript test script in Micro Focus Application Lifecycle Management, add your test script details into the VAPI-XP VBScript test script, and run the test script.

1. Navigate to the directory `IBM\SDP\almHCL\HCLOneTest\alm` in the Rational® Functional TesterHCL OneTest™ UI installation directory.

   You can use `FT_ALM_Windows.txt` file for Functional Test scripts.
2. Copy the contents of the template.
3. From Micro Focus Application Lifecycle Management, create a VAPI-XP Vbscript test script.
4. Paste the template content to the VAPI-XP Vbscript test script.
5. Enter the test details in the VAPI-XP Vbscript test script by referring to the following table:

Field	Description
Name	Required. The name of the Rational® Functional TesterHCL OneTest™ UI test.
Project Directory	Required. The fully qualified path to the Rational® Functional TesterHCL OneTest™ UI project directory. You must use '\\' or '/' as the file separator.
Script Name	Required. The name of the script to be run.
Log Format	Optional. The format of the script run logs. The options are Default, XML, HTML, text, and TPTP.
Iteration Count	Optional. The number of dataset iterations to be run.
User Arguments	Optional. Additional playback arguments, if any. If there are multiple arguments, you must use a comma to separate them.

6. Run the VAPI-XP Vbscript test script.

Results

The test result details are displayed in the Output window of Micro Focus Application Lifecycle Management.

Configuration of test runs as a GitHub Actions Workflow

You can configure test runs of test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI in a GitHub Actions Workflow. You can choose to run test resources that are in the project in the GitHub Workflow when you want to run the test resources without opening the Rational® Functional TesterHCL OneTest™ UI.

Before you configure a test run for any test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI, you must complete certain tasks. See Prerequisites to run tests in a GitHub Actions Workflow on page 481.

After you complete the prerequisite tasks, you can proceed to configure a test run in a GitHub Workflow. See Configuring a test run in a GitHub Actions Workflow on page 482.

Prerequisites to run tests in a GitHub Actions Workflow

Before you configure a test run for any test resources that are in a project repository of Rational® Functional TesterHCL OneTest™ UI, you must complete certain tasks to run the test resources in a GitHub Actions Workflow.

You must be familiar with creating projects and running test resources from Rational® Functional TesterHCL OneTest™ UI.

You must complete the following tasks in Rational® Functional TesterHCL OneTest™ UI:

- Created a test in the Functional Test perspective. See Testing in the Functional Test perspective on page 773.
- Opened and viewed the details of the test resources that you want to run in a GitHub Actions Workflow. You must make a note of the test resource details that are in Rational® Functional TesterHCL OneTest™ UI.

  You must note the following details of the test resource that are to be provided in the YAML file for the GitHub Action:
  - The complete path of the project directory that contains test assets.
  - The name of the functional test script that you want to execute.

You must complete the following tasks outside of Rational® Functional TesterHCL OneTest™ UI:

- Create an empty Git repository on any GitHub Enterprise server, if you do not already use Git repositories to store the test resources that you create. For more information about creating Git repositories, refer to Creating a new repository.

  If you are already using Git repositories to store test resources created, you can use the same repositories.

  Note: You must have the required privileges in the Git repository to create a GitHub Actions Workflow.

- Enable the GitHub Actions option in the Git repository, if it is not available as an option to use. For more information about enabling the GitHub Actions, refer to Learn about GitHub Actions.
- Set up self-hosted runners on your computer. For more information, refer to Adding self-hosted runners.

You can continue to configure a run as a GitHub action, see Configuring a test run in a GitHub Actions Workflow on page 482.

Configuring a test run in a GitHub Actions Workflow

After you complete the prerequisite tasks to run tests in a GitHub Actions Workflow, you can configure the test run in a GitHub Actions Workflow.

Before you begin

You must have completed the tasks that are mentioned in Prerequisites to run tests in a GitHub Actions Workflow on page 481.

1. Perform the following steps to open the Git repository that you use or created, which has the GitHub Actions option enabled:
    a. Open the Git repository on your GitHub Enterprise server.
    b. Create a folder with the name `.github/workflows` in the root folder of the repository.
    c. Create an empty YAML file in the `.github/workflows` folder.

    The YAML file will be used to define the GitHub Action in the Actions Workflow.
2. Perform the following steps to define the YAML file for the GitHub Action:
    a. Open the GitHub Marketplace (https://github.com/marketplace?type=) in a web browser.
    b. Search for **Rational Functional Tester GitHub ActionHCL OneTest UI GitHub Action** to run test resources that are in a Rational® Functional TesterHCL OneTest™ UI project.
    c. Open **Rational Functional Tester GitHub ActionHCL OneTest UI GitHub Action**.
    d. Read through the information provided.
    e. Copy the contents of the YAML file that is provided in the example by clicking the copy icon  .
    f. Open the YAML file that is in the `.github/workflows` folder of your repository and paste the copied contents.
    g. Refer to the following table to enter the required details in the YAML file:

Field	Description
Required	
projectDirectory	Enter the complete path to the location of the project containing the test. For example, `D:\Projects\`
scriptName	Enter the name of the test that you want to run. For example, `script1`.

Field	Description
Optional	
iterationCount	Enter the number that you want for the dataset iterations.
logFormat	Enter the format of the logs that are created when the script is run. The options are: Default, none, XML, HTML, text, and TPTP.
userArguments	Specify any arguments for the test run. For example, you can specify the playback arguments, if applicable in your test.

h. Save the YAML file.

3. Perform the following steps to set up a Runner for the GitHub Action in GitHub:

   a. Click the Settings icon .

   b. Click **Actions > Runners > New self-hosted runner**.

   c. Follow the onscreen instructions to set up a self-hosted runner on your computer.

      After the runner is installed on your computer, you must start the runner. The runner is listed in the **Runners** pane in the Git repository. The runner is in the *Idle* state when it is connected to Github and is ready to execute jobs.

      Note: You must ensure that the runner is not in the *Offline* state before you run a workflow.

4. Perform the following steps to start the GitHub Action in the Workflow:

   a. Click the **Actions** option in the Git repository.

      The GitHub Action that you configured in the YAML file for Rational® Functional TesterHCL OneTest™ UI is displayed as an **Action** under **Workflows**.

   b. Click **Rational Functional Tester GitHub ActionHCL OneTest UI GitHub Action**, and then click **Run workflow**.

Results

You have configured and run a resource that is n your project on Rational® Functional TesterHCL OneTest™ UI in a GitHub Action Workflow.

What to do next

You can perform any of the following tasks on the Git repository page by clicking the job that was completed:

- View the status of the test run by expanding the **Execute test** section.
- View the links to the test results and reports listed in the **Execute test** section.

Testing with IBM® Engineering Test Management

Rational® Functional TesterHCL OneTest™ UI can be integrated with IBM® Engineering Test Management. You can execute the functional test scripts from Rational® Functional TesterHCL OneTest™ UI.

Rational® Functional TesterHCL OneTest™ UI can be integrated with Engineering Test Management using an adapter. The functional test adapter is installed by default when you install Rational® Functional TesterHCL OneTest™ UI. After installing Rational® Functional TesterHCL OneTest™ UI, you must configure and run the adapter. The adapter receives messages from IBM® Engineering Test Management and runs functional test scripts when requested from a user through the IBM® Engineering Test Management. After test run, the adapter uploads the execution log to the IBM® Engineering Test Management server.

Test scripts are created and associated with keywords using Rational® Functional TesterHCL OneTest™ UI. These keywords are created and defined in IBM® Engineering Test Management. You can manage and execute functional test scripts and view test logs from IBM® Engineering Test Management.

You can use BuildForge to remotely install Rational® Functional TesterHCL OneTest™ UI. The administrator can use the BuildForgeScripts project provided with Rational® Functional TesterHCL OneTest™ UI to install Rational® Functional TesterHCL OneTest™ UI on a remote machine and configure the Rational® Functional TesterHCL OneTest™ UI adapter for Quality Manager integration.

The Build Forge sample scripts are located at *functional tester installation directory*`\SDP\FunctionalTester \RQMAdapter\RTLM_BuildForgeScripts. \HCLOneTest\FunctionalTester\RQMAdapter \RTLM_BuildForgeScripts.`

For information on installing Rational® Functional TesterHCL OneTest™ UI using Build Forge sample scripts see the `rtlm_rft_doc.html` document at *functional tester installation directory*`\SDP\FunctionalTester \RQMAdapter\RTLM_BuildForgeScripts\Docs\HCLOneTest\FunctionalTester\RQMAdapter \RTLM_BuildForgeScripts\Docs`.

If a functional test script depends on external JARs, and you want to run the test script from IBM® Engineering Test Management, you must place the test script in the Rational® Functional TesterHCL OneTest™ UI customization folder. For details, see https://jazz.net/forum/questions/67674/classnotdeffexception-whenn-running-a-rft-script-from-rqm.

Configuring and running Rational® Functional TesterHCL OneTest™ UI adapter for IBM® Engineering Test Management

You can integrate Rational® Functional TesterHCL OneTest™ UI with Engineering Test Management by using an adapter. The adapter is available in the Rational® Functional TesterHCL OneTest™ UI installation directory. You can run the tests that you create by using Rational® Functional TesterHCL OneTest™ UI in Engineering Test Management after you configure and start the adapter.

Before you begin

- If you are a Engineering Test Management user, you must have appropriate permissions to use the Rational® Functional TesterHCL OneTest™ UI adapter for Engineering Test Management.
- You must have a Connector client access license or a license that supersedes the Connector client access license. For more information, refer to Client access license management in the Engineering Lifecycle Management Solution knowledge center.
- You must have modified the hosts file and map the host name of the Engineering Test Management server to its IP address if the host name of your Engineering Test Management server is not registered with the DNS server.



Important: You must configure and start the adapter through the command-line interface to connect the adapter to the Engineering Test Management server through a proxy server. For information, see Related Links.

1. Reconfigure the JRE path if you are using Kerberos for authentication:
   a. Modify the JRE path to use the Oracle Java path as follows in the `eclipse.ini` file in the `C:\Program Files\IBM\SDPC:\Program Files\HCL\SDP` directory:

      Change `-vm jdk/jre/bin/j9vm/jvm.dll` to

      `-vm C:\Program Files\Java\jre1.8.0_331\bin\server\jvm.dll`.

   b. Navigate to the `FunctionalTester> RQMAdapter` directory in the Rational® Functional TesterHCL OneTest™ UI installation directory.
   c. Replace *%RFT_ARG_JRE%* with the Oracle Java path on your system in the following line of the `configureadapter.bat` file:

      ```
      start "" %RFT_ARG_JRE% -classpath %RFT_INSTALL_DIR%\rational_ft.jar
      com.rational.test.ft.RQMAdapterConfigure %*.start "" %RFT_ARG_JRE% -classpath %RFT_INSTALL_DIR
      %\hcl_ft.jar com.rational.test.ft.RQMAdapterConfigure %*.
      ```

      For example,

      ```
      start "" "C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe" -classpath %RFT_INSTALL_DIR
      %\rational_ft.jar com.rational.test.ft.RQMAdapterConfigure %*start "" "C:\Program
      Files\Java\jre1.8.0_321\bin\javaw.exe" -classpath %RFT_INSTALL_DIR%\hcl_ft.jar
      com.rational.test.ft.RQMAdapterConfigure %*
      ```

2. Run `configureadapter.bat` in the Windows operating systems or `configureadapter.sh` in the Linux operating systems.

   The **IBM Rational Functional Tester AdapterHCL OneTest UI Adapter** dialog box is displayed.

3. Complete the following steps from the **Connection Information** tab:

a. Enter the server URL of Engineering Test Management in the **Server URL** field.



Note: The URL that you enter in the **Server URL** field must match with the public URI of the Engineering Test Management server. You can view the public URI on the administrator page of the Engineering Test Management server.

b. Select the mode that you want to use to authenticate the connection with the Engineering Test Management server from the **Authentication Type** drop-down list.

You can select any of the options in the following table, and then complete the steps for that option:

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Username and Password</td><td>You can establish the connection with the Engineering Test Management server by using the user ID and password.

Complete the following actions:

<table>
<tr><th>Field</th><th>Action</th></tr>
<tr><td>User ID</td><td>Enter the user ID that you use to log in to Engineering Test Management</td></tr>
<tr><td>Password</td><td>Enter the password that you use to log in to Engineering Test Management.</td></tr>
<tr><td>Project area</td><td>Select the required project area in the Engineering Test Management server from the drop-down list.

  Note: This field is populated with the project names that are on the Engineering Test Management server and the project names are displayed after the connection is established.

  Important: You must select the project from the drop-down list and not type the project name.</td></tr>
<tr><td>Save password</td><td>Select this option to save your Engineering Test Management password.</td></tr>
</table></td></tr>
<tr><td>KERBEROS</td><td>You can establish the connection with the Engineering Test Management server by using the Kerberos initialization file (<code>krb5.ini</code>).</td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td></td><td>Complete the following actions:
<table><tr><th>Field</th><th>Action</th></tr><tr><td>Configuration File</td><td>Click Browse and select the <code>krb5.ini</code> file.</td></tr><tr><td>Project area</td><td>Select the required project area in the Engineering Test Management server from the drop-down list.

Note: This field is populated with the project names that are on the Engineering Test Management server and the project names are displayed after the connection is established.

Important: You must select the project from the drop-down list and not type the project name.</td></tr><tr><td>Save password</td><td>Select this option to save your Engineering Test Management password.</td></tr></table></td></tr>
<tr><td>SSLCERT</td><td>You can establish the connection with the Engineering Test Management server by using an SSL certificate.

Complete the following actions:
<table><tr><th>Field</th><th>Action</th></tr><tr><td>Certificate Location</td><td>Click Browse and select the SSL certificate.

Note: The certificate must be a .p12 file.</td></tr><tr><td>Password</td><td>Enter the password that you use to log in to Engineering Test Management.</td></tr><tr><td>Project area</td><td>Select the required project area in the Engineering Test Management server from the drop-down list.

Note: This field is populated with the project names that are on the Engineering Test Management server</td></tr></table></td></tr>
</table>

Option	Description
	Field / **Action** and the project names are displayed after the connection is established. **Important:** You must select the project from the drop-down list and not type the project name.
SMARTCARD	You can establish the connection with the Engineering Test Management server by using a smart card. **Notes:** ▪ Smart card authentication is only available in Windows operating systems. ▪ The computer must be connected to the smart card reader before you select this option. You must also insert the smart card into the smart card reader. Rational® Functional TesterHCL OneTest™ UI reads the aliases from the smart card and prompts for the smart card PIN when the adapter connects to Engineering Test Management. Complete the following actions: **Field** / **Action** Certificate Location: Select the smart card certificate from the drop-down list. Project area: Select the required project area in the Engineering Test Management server from the drop-down list. **Note:** This field is populated with the project names that are on the Engineering Test Management server

Option	Description	
	Field	**Action**
		and the project names are displayed after the connection is established. **Important:** You must select the project from the drop-down list and not type the project name.

c. Enter the adapter name in the **Adapter Name** field.

d. Click **Apply**.

4. Click **Start Adapter** to start the connection between Rational® Functional TesterHCL OneTest™ UI and Engineering Test Management.

Note: You can view the status of the connection by clicking the **Adapter Console** tab. You can stop the connection by clicking **Stop Adapter**.

Related information

Using the adapter to connect to IBM Engineering Test Management through a proxy server on page 491

Configuring and running the adapter using the command-line interface

Integration with Rational® Functional TesterHCL OneTest™ UI is implemented through the use of an adapter. The functional test adapter is installed by default when you install Rational® Functional TesterHCL OneTest™ UI. You can configure and start the adapter using the command-line interface.

Before you begin

If you are a Engineering Test Management user, ensure that you have the appropriate permissions, including execute permissions, to access the Rational® Functional TesterHCL OneTest™ UI adapter for Engineering Test Management. If you do not have administrator rights then use the **Connector Client Access** license. You can obtain this license from the Engineering Test Management administrator.

If the host name of the Engineering Test Management server is not registered with the DNS server, modify the hosts file and map the host name of the Engineering Test Management server to its IP address.

Note: If the adapter connects to the Engineering Test Management server through a proxy server, configure and start the adapter through the command-line interface using the instructions in Configuring and running the adapter through a proxy server on page 491.

1. At the prompt, type the following command-line arguments to run the `configureadapter.bat` configuration batch file:`configureadapter.bat -repository https://<rqmserver>:<port>/qm -user <userid> -password <password> -adapter <adapter name> [-projectArea <projectArea>] [-configfile <configuration file>] [-interactive <true/false>]`
   where:
   - `<rqmserver>` is the host name of the Engineering Test Management server.
   - `<port>` is the port where the Engineering Test Management server is running.
   - `<userid>` is a registered user ID in Engineering Test Management that has the license to run the adapter.
   - `<password>` is the password for the user ID that was used.
   - `<adapter name>` is the name you give your adapter and by which the Engineering Test Management web UI will identify this instance of the adapter.
   - `<projectArea>` is the name or alias of the project to which you are logging on. If the name contains spaces, enclose the name in double quotation marks.
   - `<configuration file>` is the file from which the settings for this adapter must be read. The adapter name is mandatory when you use this option.
   - `[-interactive <true/false>]`: If you set this value to `true`, the Rational® Functional TesterHCL OneTest™ UI Adapter dialog box opens if you have not specified values for any of the command-line arguments. You can then specify the missing value in the Rational® Functional TesterHCL OneTest™ UI Adapter dialog box.

   Note: If the Engineering Test Management server is renamed, complete these steps:
   a. If you have an entry for the Engineering Test Management server in the hosts file, update it with the new server name.
   b. Update the `<rqmserver>` argument with the new server name.
   c. Configure the adapter again to point to the new URL

2. Start the adapter by running the `startadapter.bat` batch file, which is in the same location as the `configureadapter.bat` file. Type the following command-line arguments at the prompt: `startadapter.bat -repository https://<rqmserver>:<port>/qm -user <userid> -password <password> -adapter <adapter name> -projectArea <project area> [-configfile <configuration file>]`
   where:
   - `<rqmserver>` is the host name of the Engineering Test Management server.
   - `<port>` is the port where the Engineering Test Management server is running.
   - `<userid>` is a registered user ID in Engineering Test Management that has the license to run the adapter.
   - `<password>` is the password of the user ID used

- <adapter name> is the name you give your adapter and by which the Engineering Test Management web UI will identify this instance of the adapter.
- <projectArea> is the name or alias of the project to which you are logging on. If the name contains spaces, enclose the name in double quotation marks.
- <configuration file> is the file from which the settings for this adapter must be read. The adapter name is mandatory when you use this option.

Using the adapter to connect to IBM® Engineering Test Management through a proxy server

You can use the adapter to connect to the Engineering Test Management server through a proxy server. You must configure the credentials for the proxy server from the command-line interface.

Before you begin

If you are a Engineering Test Management user, ensure that you have the appropriate permissions, including execute permissions, to access the Rational® Functional TesterHCL OneTest™ UI adapter for Engineering Test Management. If you do not have administrator rights then use the **Connector Client Access** license. You can obtain this license from the Engineering Test Management administrator.

If the host name of the Engineering Test Management server is not registered with the DNS server, modify the hosts file and map the host name of the Engineering Test Management server to its IP address. Additionally, map the host name of the proxy server through which the adapter connects to the Engineering Test Management server to the proxy IP address.

About this task

If the adapter connects to the Engineering Test Management server through a proxy server, you must specify the logon credentials for the proxy server for authentication purposes. Use the command-line interface to provide the proxy server logon credentials.

1. At the prompt, type the following command-line arguments to run the configureadapter.bat configuration batch file:

```
configureadapter.bat -repository https://<rqmserver>:<port>/qm -user <userid> -password <password> -adapter <adapter name> [-projectArea <projectArea>] -proxy <proxy server name> -proxyPort <proxy port number> -proxyUser <proxy server userid> -proxyPassword <proxy server password> [-configfile <configuration file>] [-interactive <true/false>]
```

   where:
   - <rqmserver> is the host name of the Engineering Test Management server.
   - <port> is the port where the Engineering Test Management server is running.
   - <userid> is a registered user ID in Engineering Test Management that has the license to run the adapter
   - <password> is the password for the user ID that was used.
   - <adapter name> is the name you give your adapter and by which the Engineering Test Management web UI will identify this instance of the adapter.

- <projectArea> is the name or alias of the project to which you are logging on. If the name contains spaces, enclose the name in double quotation marks.
- <proxy server name> is the host name of the proxy server through which the adapter connects to the Engineering Test Management server.
- <proxy port number> is the port where the proxy server is running.
- <proxy server userid> is the user ID that is used to authenticate the proxy server.
- <proxy server password> is the password for the proxy server authentication user ID
- <configuration file> is the file from which the settings for this adapter must be read. The adapter name is mandatory when you use this option.
- [-interactive <true/false>]: If you set this value to true, the Rational® Functional TesterHCL OneTest™ UI Adapter dialog box opens if you have not specified values for any of the command-line arguments. You can then specify the missing value in the Rational® Functional TesterHCL OneTest™ UI Adapter dialog box.

Note: If the Engineering Test Management server is renamed, complete these steps:

a. If you have an entry for the Engineering Test Management server in the hosts file, update it with the new server name.
b. Update the <rqmserver> argument with the new server name.
c. Configure the adapter again to point to the new URL

2. Start the adapter by running the startadapter.bat batch file, which is in the same location as the configureadapter.bat file. Type the following command-line arguments at the prompt: `startadapter.bat -repository https://<rqmserver>:<port>/qm -user <userid> -password <password> -adapter <adapter name> -projectArea <project area> -proxy <proxy server name> -proxyPort <proxy port number> -proxyUser <proxy server userid> -proxyPassword <proxy server password> [-configfile <configuration file>]`
where:
   - <rqmserver> is the host name of the Engineering Test Management server.
   - <port> is the port where the Engineering Test Management server is running.
   - <userid> is a registered user ID in Engineering Test Management that has the license to run the adapter.
   - <password> is the password for the user ID that was used
   - <adapter name> is the name you give your adapter and by which the Engineering Test Management web UI will identify this instance of the adapter.
   - <projectArea> is the name or alias of the project to which you are logging on. If the name contains spaces, enclose the name in double quotation marks.
   - <proxy server name> is the host name of the proxy server through which the adapter connects to the Engineering Test Management server.
   - <proxy port number> is the port where the proxy server is running.
   - <proxy server userid> is the user ID that is used to authenticate the proxy server.

- `<proxy server password>` is the password for the proxy server authentication user ID
- `<configuration file>` is the file from which the settings for this adapter must be read. The adapter name is mandatory when you use this option.

Note:

- For the adapter to connect to the Engineering Test Management server through the proxy server, you must specify the proxy server name, port number and user ID.
- If the proxy server is renamed, complete these steps:
  a. If you have an entry for the proxy server in the hosts file, update it with the new proxy server name.
  b. Update the `<proxy server name>` argument with the new proxy server name.

Handling inputs from IBM® Engineering Test Management in Rational® Functional TesterHCL OneTest™ UI scripts

If you use IBM® Engineering Test Management to execute functional test scripts, you can make Engineering Test Management details available to the functional test script to be used during playback.

Engineering Test Management details could include script arguments and dataset iterations. You can use one of these methods to make Engineering Test Management details available to functional test scripts, depending on your requirements:

- Pass Engineering Test Management details and arguments while running a specific functional test script. These arguments are specific and local to that script alone. For information about using this method, see IBM Engineering Test Management script arguments on page 493.
- Pass details from Engineering Test Management at a test script, test case or test suite level. You can use execution variables to pass details at any or all of these levels. For information about this method, see IBM Engineering Test Management execution variables on page 495.

Related information

Working with keywords in IBM Rational Functional TesterHCL OneTest UI on page 497

Testing with IBM Engineering Test Management on page 484

Configuring and running Rational Functional TesterHCL OneTest UI adapter for IBM Engineering Test Management on page 484

Configuring and running the adapter using the command-line interface on page 489

Viewing keywords created in Engineering Test Management on page 497

IBM® Engineering Test Management script arguments

You can pass arguments from IBM® Engineering Test Management while running an IBM® Rational® Functional TesterHCL OneTest™ UI script.

You can pass the following types of arguments while running functional test scripts from Engineering Test Management:

- **Script arguments**: These arguments are available to a script writer within the script as arguments to the testMain() method.
- **Execution arguments**: These arguments govern playback. If Rational® Functional TesterHCL OneTest™ UI test scripts that run from Engineering Test Management have a dependency on a third-party library, you must consider the following points:
  - You must use the `-projectpath` or `-classpath` command line argument to specify a third-party library. For example: `-projectpath C:\temp\myjar.jar`. See Command line interface on page 1726.

    Note: Command line arguments such as `-enable`, `-inspector`, or `-appconfig` that are not applicable for Engineering Test Management is not considered during playback.

  - You can pass java properties along with execution arguments. For example, `-Dmyprop=value`. You can also pass multiple java properties that are separated by space along with execution arguments. For example, `-Dmyprop1=value1 -Dmyprop2=value2 -projectpath C:\tmp\myjar.jar`.
- **dataset iteration**: If there is a dataset associated with a script, you can pass the number of times the scripts must run by accessing records from the dataset.

Additionally, you can pass execution task details of Engineering Test Management to a functional test script so that information about the Engineering Test Management test cases is accessed during script execution.

To access execution task details within the testMain() method in a functional test script, you must provide the path to the XML file that describes the Engineering Test Management execution task.

In the functional test script, you must add `String path = System.getProperty("rqm.task")` to get the path of the XML file.

Refer to the *XML Representation of an adapter Task* section on the http://open-services.net/bin/view/Main/QmExecutionAdapter web page for information about the schema of the execution task that is represented in this file.

This file points to other resources for other Engineering Test Management assets such as test cases. You can find the schemas for the other resources on the same site. By using the data in this XML file, you can communicate with Engineering Test Management.

Note: The details Engineering Test Management that you pass using this method are specific and local to the script. If you want to specify details from the Engineering Test Management test script, test case, or test

suite levels, you use execution variables. See IBM Engineering Test Management execution variables on page 495.

IBM Engineering Test Management execution variables

If you use IBM® Engineering Test Management to run functional test scripts, you can create variables for parameters within a functional test script that is associated with a Engineering Test Management test suite, test case, or test script. These variables, known as execution variables, can be passed to the functional test script to be used during playback.

When the Engineering Test Management test case is run, the execution variables that have been created are obtained by the functional test adapter and passed to the functional test script to be used on playback. Values for the execution variables can be supplied from the command-line prompt, a text file or worksheet, or from the associated test suite, test case, or test script. The details of the execution variables that are passed to the functional test script are displayed on the Execution Variable tab on the Execution Results page in Engineering Test Management.

For example, to log values for data such as user name and password in a functional test script, you can create variables for user name and password, either in Engineering Test Management or in the functional test script. When the associated test case is run in Engineering Test Management, the functional test adapter obtains the user name and password variables and provides them to the functional test script on playback. Values for the user name and password execution variables can be specified at the command line prompt, in a comma-separated values (CSV) file or text file. The values can also be specified in the associated test suite, test case or test script. Details of the user name and password variables are displayed on the Execution Variable tab on the Execution Results page in Engineering Test Management. The values for user name and password are displayed in the log.

Note: By using execution variables, you can pass parameters at the test script, test case or test suite levels. To pass details only to a specific script, you can also use arguments to the testMain() method in the script. For more information, see IBM Engineering Test Management script arguments on page 493.

Note: The execution variables feature is available only with Engineering Test Management, version 3.0.1.

Reading variables

When a Engineering Test Management test case or test script is run, the functional test adapter reads the execution variables that were created for the test case or script and passes the variables to the functional test script playback engine. The functional test playback engine provides the variables to the functional test script that is associated with the Engineering Test Management test case or script. On playback, the functional test script uses the variables and obtains values for the variables.

You must modify the functional test script to enable it to read the Engineering Test Management execution variables during playback. This code is provided in the IVariablesManager API:

```
IVariablesManager vm=getVariablesManager()
```

You must modify functional test script to enable it to read the parameter names for the Engineering Test Management execution variables during playback. This code is provided in the IParameter API:

```
IParameter name = vm.getInputParameter("name")
```

Creating variables

You can create execution variables within the associated functional test script.

To create execution variables within the functional test script, this code is provided in the IVariablesManager API:

```
IVariable <var name> = vm.createOutputVariable("<var name>", "<var value>");
```

Modifying variables

You can modify execution variables created in Engineering Test Management either in the test suite, test case, or test script. You can modify execution variables created in the functional test script.

Note: Execution variables created in Engineering Test Management cannot be modified in the functional test script, but can only be read.

To modify execution variables that were created earlier in the functional test script, this code is provided in the IVariable API and the IVariablesManager API:

```
<var name>.setValue("<new var value>");
vm.setOutputVariable(<var name>);
```

Test cases in a test suite

For test cases in a test suite, the functional test output variables for a test case, if any, are provided as input variables for the next test case in the suite.

Rational® Functional TesterHCL OneTest™ UI in stand-alone mode

When Rational® Functional TesterHCL OneTest™ UI is in stand-alone mode without Engineering Test Management, the execution variables can be read from the command line, or from a text file or worksheet.

Use a -var extension to enable Rational® Functional TesterHCL OneTest™ UI to read execution variables and their values from the command line. For example, type:

```
<playbackcmd> -var "username=user1;password=pass1"
```

Use the -varfile extension to enable Rational® Functional TesterHCL OneTest™ UI to read execution variables and their values from a text file or worksheet, for example:

```
<playbackcmd> -varfile <file containing values>
```

In the text file or worksheet, each variable name and value pair must be on a new line. If comma -separated values are provided, they are treated as a single value.

Working with keywords

In Rational® Functional TesterHCL OneTest™ UI, you can associate functional test scripts with the keywords that are created in IBM® Engineering Test Management.

Working with keywords in IBM® Rational® Functional TesterHCL OneTest™ UI

Keywords are defined and created in IBM® Rational® Quality Manager. In IBM® Rational® Functional TesterHCL OneTest™ UI, you can record a functional test script and then associate it with the keywords that are created in Rational® Quality Manager.

Learn more about keywords: Keywords are defined and created in Rational® Quality Manager. A keyword is a statement or group of statements that you can reuse in other test scripts. Keywords are typically composed of script steps that reflect reusable processes. You can automate keywords through the use of functional test scripts. For more information about keywords, see the Rational® Quality Manager Knowledge Center.

To work with keywords, you must log on to Rational® Quality Manager server. The keywords created in Rational® Quality Manager are displayed in the Rational® Functional TesterHCL OneTest™ UI Keyword View. You can record a functional test script and associate it with an keyword.

After you associate the functional test scripts to the appropriate keywords, you can play back the functional test scripts that are associated with the keywords from Rational® Quality Manager.

Viewing keywords created in Engineering Test Management

In Rational® Functional TesterHCL OneTest™ UI, you can record and then associate functional test scripts with the keywords that are created in IBM® Rational® Quality Manager. You can log on to the Rational® Quality Manager to view keywords. You can play back the functional test script from Engineering Test Management.

1. In the Keyword View, click **Logon to RQM**  .
2. In the **Logon to Rational Quality Manager** dialog box, type the URL in the RQM repository field. For example: If Rational® Quality Manager is running in the local computer, type `https://localhost:9443/qm` in the RQM repository field.

   Note: If the Rational® Quality Manager server is renamed, make sure that you update the URL in the **RQM repository** field with the new server name.

3. In **Authentication Mode**, specify the authentication type to connect to Rational® Quality Manager and click **Finish**.

- User name and Password - Specify the user ID and password of the Rational® Quality Manager user account.
- Kerberos - Select the kerberos .ini file. The file is automatically created when you set up Kerberos. Typically, on Windows, the file would be located at `c:\windows\krb5.ini`. The file name and the location would change for different operating systems.
- SSL Cert - Specify the location of the SSL certificate and optionally, the password. The certificate should be a .p12 file.
- Smart Card - Select the smart card certificate alias. You must ensure that the workbench machine is connected to the smart card reader and that the smart card is inserted into the reader. The product reads the aliases from the smart card and prompts for the smart card PIN whenever the adapter connects to Rational® Quality Manager.

4. Click **Get Keywords**  . The connected projects keyword list is displayed in the keyword view.

   a. To select a project area, click **ProjectArea** list.
   Result
   Only keywords pertaining to the selected project area are displayed.

   b. Click **Get Keywords**.

   a. **Optional:** To search for keywords that have a specific tag, type the tag in the **Search by tag** field.
   b. **Optional:** Click **Get Keywords**.

   Result
   If a large number of keywords are displayed, they are organized into pages of 50 each. Use the navigation buttons on the toolbar to move between pages.

Associating functional test scripts with the keywords

You can automate an IBM® Rational® Quality Manager keyword by associating it with an IBM® Rational® Functional TesterHCL OneTest™ UI script. You can either associate an existing functional test script with a given keyword, or record a new script against the keyword and then associate the script. You can see the steps of a keyword when you associate an automated script. After a keyword is automated, you can open and execute the automated script.

Before you begin

The Keyword View must be displayed in the Functional Test perspective to associate functional test scripts with keywords. To display the Keyword View, click **Window > Show View**. Select **Keyword View** from the list, and click **OK**.

Recording a new functional test script for a keyword

1. Click the keyword from the keyword view list to record a script and associate it with a functional test script.
2. Right-click the keyword, and click **Record** .
3. Type a name for the functional test to be recorded in **Script name**.
   By default, the keyword name is used as the functional test script name.
4. Click **Finish**.

Result

The Recording monitor opens and the recording starts. The keyword description is displayed in the recording monitor. For information on recording, see the related topic on Recording a script.

Associating an existing functional test script with a keyword

1. Click the keyword from the keyword view list to associate it with a functional test script.
2. Right-click the keyword, and click **Associate with FT script**  .

   Result

   The **Select Script** dialog box lists the existing functional test scripts for the functional test projects.
3. Select the functional test script that you want to associate from the list and click **OK**.

   Result

   The **Configure shared location** wizard is displayed when you associate a functional test script with the keyword for the first time.

   Note: Only Java test scripts are listed. Simplified scripts in the project, if any, are represented by their equivalent Java test script.

4. The **Configure shared location** wizard specifies the shared location of the functional test project. Type the path to the parent directory of the functional test project in the **Shared location** field. You can also select the shared location from the **Shared Locations already configured in RQM** list box.
5. Click **Finish**.

   Result

   If you click **Cancel**, the project is not configured to use the shared location. In this case Rational® Quality Manager uses the absolute path of the current project to execute the test script.
6. Copy the project that contains the keyword-associated script to the specified shared location manually. For Rational® Quality Manager to run the script, the project with the script must be present in the shared location.

   Result

   When you associate a functional test script with a keyword, the script becomes the default script for the keyword. Only a single functional test script can be associated with a keyword. When you run the keyword from Rational® Quality Manager, the test script associated with the keyword is executed.

Running functional test scripts from Rational Quality Manager

You can either reference external resources by accessing a shared location or a local test machine. During execution of the test scripts, the test resources are copied to the test machine if it is a shared location. However, when you access resources in a local test machine, you are accessing resources on the path that you use.

1. In Rational® Quality Manager, create a test case.
2. Create a test script name and associate an existing test script with the test case in Rational® Quality Manager.
3. Execute the test case and view the results of the test execution after the playback. For information, refer the Rational® Quality Manager information center.

Related information

ShowMe

Keyword View

The Rational® Functional TesterHCL OneTest™ UI Keyword View displays the keywords. This view is displayed in the right pane of the Functional Test perspective.

The following menu options are available when you right-click a keyword in the keyword view:

Refresh Steps

Lists the steps associated with a keyword.

Record Test

Records a functional test script for the selected keyword.

Associate Test

Associates a functional test script with the keyword. Multiple functional test scripts can be associated with a keyword.

Show Associated Tests

Shows all the functional test scripts associated with a keyword.

To view the keyword list:

- If the Keyword View is not displayed in the Functional Test perspective, click **Window > Show View**. Select **Keyword View** from the list and click **OK**.

To select a project area:

- To select a project, click **ProjectArea** list. Only keywords pertaining to the selected project area are displayed.

Troubleshooting: Unable to open unified reports

When you try to open the unified report that is published to the Engineering Test Management after the test run, the report fails to open in the browser.

Exemple

Solution:

Perform the following steps to open the unified report in a browser:

1. Log in to Engineering Test Management.
2. Click the **Administration** icon  in the upper-right corner of the header, and then select **Manage Project Properties**.

3. Click **Test Result Preference** in the **Properties** tab.
4. Select the **Open attachments in test results directly, without prompting users to open or save them** checkbox.

   Note: This is a one-time setup to enable the option to open reports.

5. Click the unified report link to open the report in a browser.

Testing with IBM® Engineering Test Management in the Visual Studio IDE

Rational® Functional TesterHCL OneTest™ UI in the Visual Studio environment supports integration with IBM® Engineering Test Management.

Exemple

- Testing with IBM Engineering Test Management on page 484
- Configuring and running Rational Functional TesterHCL OneTest UI adapter for IBM Engineering Test Management on page 484
- Configuring and running the adapter using the command-line interface on page 489
- Using the adapter to connect to IBM Engineering Test Management through a proxy server on page 491
- IBM Engineering Test Management execution variables on page 495

Testing with Jenkins

You can use the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test to run tests on a Jenkins server.

To automate testing with Jenkins, you must configure Jenkins primary server and Jenkins secondary server. This configuration provides a single Jenkins installation on the Jenkins primary server to host multiple Jenkins secondary server for building and running tests. For more information about the Jenkins primary and secondary server relationship, refer to the Jenkins documentation.

You must install the required version of the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin on the Jenkins primary server, and install Rational® Functional TesterHCL OneTest™ UI on the Jenkins secondary server, where you create tests.

You can use either the Jenkins **Freestyle** project or the **Pipeline** project to run test assets from Jenkins. With **Freestyle** project, you can create a build step from the Jenkins UI to run the test assets. Whereas the **Pipeline** project uses a simple text scripts based on the Groovy programming language. You can define pipeline scripts during the configuration of the **Pipeline** project by using any of the following methods:

- **Pipeline script**: In this method, you can directly enter a script in the field provided within the Jenkins UI or you can select a sample **Pipeline script** from the drop-down list.
- **Pipeline script from SCM**: In this method, you can create a script with a Groovy editor and then commit the script file into the Git repository.

For more information about the Jenkins Pipeline, refer to the Jenkins documentation.

Refer to the following topics to learn more about integrating Jenkins with Rational® Functional TesterHCL OneTest™ UIin the Functional Test perspective:

Environment variables

You can add environment variables on the Jenkins server to run the Jenkins build by referring to environment variables.

You can add an environment variable on the Jenkins server by navigating to **Manage Jenkins > Configure System > Global properties**. You can enter the variable name by using any of the following methods for the corresponding text fields in the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test** step:

- Use the dollar sign ($) followed by the variable name.

  For example, `$workspace`

- Use the dollar sign ($) followed by the variable name between braces.

  For example, `${workspace}`

The IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test uses the actual value while running the job.

For example, if you add the environment variable named `workspace` with the value `C:\Users\IBM\workspace1C:\Users\HCL\workspace1`, then you can use `$workspace` or `${workspace}` as input to the **Workspace** field when running tests. During the run time, `$workspace` or `${workspace}` is substituted with its corresponding value `C:\Users\IBM\workspace1C:\Users\HCL\workspace1`.

Build parameters for Jenkins

You can define the build parameters when you configure either **Freestyle** or **Pipeline** projects. You must map the build parameters with the parameters of **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test**. When you use the build parameters, Jenkins prompts you to enter a value for the defined parameter when you run the Jenkins job. Each parameter has a name and value.

Jenkins supports various types of build parameters. You can choose the type of parameter based on your requirements. Some of the commonly used types of parameters are as follows:

- Boolean
- Choice
- Credentials
- File
- String
- Multi-line String
- Password

After you add the parameter, the name that you provided for the parameter must be entered in the corresponding field in the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test** step in the following format:

`${name_of_the_parameter}`

For example, `${workspace}`

Each time when you run a test from Jenkins, you might want to specify the dataset iteration count. In such a case, you can add a **String Parameter** by selecting the **This project is parametrized** option from the **General** tab when you configure the project.

You can then provide a name for the string parameter as *Count* in the **Name** field. You can also provide a default value to be used in the **Default Value** field and describe the parameter in the **Description** field. These are optional fields.

When you provide the details of the test run in the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test** step, you can enter the value for the **Iteration Count** field as `${Count}`.

When you run the Jenkins job, the *Count* parameter prompts for you to specify the number of iterations. You can provide the number of dataset iterations to be used for the test run. You must then click **Build** to run the test.

Related information

Configuring the Freestyle project on page 505

Creating a pipeline script from Jenkins on page 507

Running tests from Jenkins on page 510

Task flows for running test assets from Jenkins

You can perform certain tasks to run test assets from the Jenkins **Freestyle** project or the **Pipeline** project.

The following table lists the task flows for running test assets from the Jenkins **Freestyle** project:

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 504
Configure the **Freestyle** project.	Configuring the Freestyle project on page 505
Run Rational® Functional TesterHCL OneTest™ UI functional tests on Jenkins.	Running tests from Jenkins on page 510

You can perform the following tasks to run test assets from the Jenkins **Pipeline** project by using the sample script:

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 504
Create a pipeline script from the Jenkins UI.	Creating a pipeline script from Jenkins on page 507
Configure the **Pipeline** project.	Configuring the Pipeline project by using the sample script on page 508
Run Rational® Functional TesterHCL OneTest™ UI functional tests on Jenkins.	Running tests from Jenkins on page 510

You can perform the following tasks to run test assets from the Jenkins **Pipeline** project by using the script from Source Code Management (SCM):

Tasks	More information
Install the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test Jenkins plugin.	Installing the plugin on the Jenkins primary server on page 504
Create a pipeline script and commit the script to the Git repository.	For more information, refer to the Jenkins documentation.
Configure the **Pipeline** project.	Configuring the Pipeline project by using the script from SCM on page 509
Run Rational® Functional TesterHCL OneTest™ UI functional tests on Jenkins.	Running tests from Jenkins on page 510

Installing the plugin on the Jenkins primary server

You must install the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test Jenkins plugin to run test assets from the Jenkins server.

Before you begin

You must have completed the following tasks:

- Verified that you have a Jenkins primary server and secondary server.
- Downloaded the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test Jenkins plugin `RFT-Jenkins-FunctionalTest-7.1HOT-UI-Jenkins-FunctionalTest-7.1` from the IBM WebSphere, Liberty & DevOps Community portalHCL® License & Delivery portal.

1. Log in to the Jenkins server.
   Result
   The Jenkins dashboard is displayed.
2. Click **Manage Jenkins > Manage plugins**, and then click **Advanced** tab.
3. Click **Choose File** and then locate and open the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin.
4. Click **Upload**.
   Result
   The IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test is displayed in the **Installed** tab.
5. Perform the following steps to provide Random TCP Ports for Java™ Network Launch Protocol (JNLP) agents:
   a. Click **Manage Jenkins** from the Jenkins dashboard.
   b. Click **Configure Global Security** from the **Security** section.
   c. Click **Random** from the **Agents** section.
   d. Click **Save** to save and apply the changes.

Results

You have installed the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin on the Jenkins primary server.

What to do next

You can run the test from the Jenkins server. See Running tests from Jenkins on page 510.

Configuring the Freestyle project

You must configure a **Freestyle** project to add a build step, and then run test assets from Jenkins.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 504.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.
- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.
- Created a Jenkins **Freestyle** project.

About this task

When you create a **Freestyle** project in the Jenkins server, you must select the **Restrict where this project can be run** checkbox and enter the name of the labels that you provided during the creation of Agent in the **Label Expression** field.

1. Open the Jenkins **Freestyle** project, and then click **Configure**.
2. Click the **Build** tab, and then click **Add build step**.
3. Select the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test** option from the drop-down list.
4. Provide the details about the test run for the fields in the following table:

Field	Description
Name	Required. The name of the Rational® Functional TesterHCL OneTest™ UI test.
Project Directory	Required. The fully qualified path to the Rational® Functional TesterHCL OneTest™ UI project directory. You must use '\\' or '/' as the file separator.
Script Name	Required. The name of the script to be run.
Log Format	Optional. The format of the script run logs. The avaiable options are as follows: ◦ Default ◦ none ◦ json ◦ xml ◦ html ◦ TPTP ◦ Text
Iteration Count	Optional. The number of dataset iterations to be run.
User Arguments	Optional. Additional playback arguments, if any. For multiple arguments, you must enclose each argument within double quotation marks and separate the arguments by providing a space between them. For example, "password" "7891230" "20".
Project Dependencies	Optional. The complete path to the project that your test depends for the run. If there are multiple projects, you must separate each project path with a semicolon.

5. **Optional:** Click **Add build step** again, and provide details for the next test to run multiple tests under the same job.
6. Click **Save**.

Results

You have configured the **Freestyle** project by adding the build step.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 510.

Creating a pipeline script from Jenkins

You must create a pipeline script to run test assets from the Jenkins **Pipeline** project.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 504.
- Created a Jenkins **Pipeline** project.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Click **Pipeline Syntax** to generate a pipeline script.
4. Select the **step: General Build Step** option from the drop-down list in the **Sample Step** field.
5. Select the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI - Functional Test** option from the drop-down list in the **Build Step** field.
6. Provide the details about the test run for the fields in the following table:

Field	Description
Name	Required. The name of the Rational® Functional TesterHCL OneTest™ UI test.
Project Directory	Required. The fully qualified path to the Rational® Functional TesterHCL OneTest™ UI project directory. You must use '\\' or '/' as the file separator.
Script Name	Required. The name of the script to be run.
Log Format	Optional. The format of the script run logs. The avaiable options are as follows: ◦ Default ◦ none ◦ json ◦ xml ◦ html ◦ TPTP ◦ Text
Iteration Count	Optional. The number of dataset iterations to be run.
User Arguments	Optional. Additional playback arguments, if any. For multiple arguments, you must enclose each argument within double quotation marks and sep-

Field	Description
	arate the arguments by providing a space between them. For example, "password" "7891230" "20".
Project Dependencies	Optional. The complete path to the project that your test depends for the run. If there are multiple projects, you must separate each project path with a semicolon.

7. Click **Generate Pipeline Script** and copy the script that you created for the build step to use it in the sample script.

Note:

You need this generated script to enter in step 5.c on page 509 in the following topic:

Configuring the Pipeline project by using the sample script

Results

You have created the pipeline script.

What to do next

You can configure the **Pipeline** project by using the **Pipeline script** option. See Configuring the Pipeline project by using the sample script on page 508.

Configuring the Pipeline project by using the sample script

You can configure a **Pipeline** project directly from the Jenkins UI by using the sample pipeline scripts to run test assets from Jenkins.

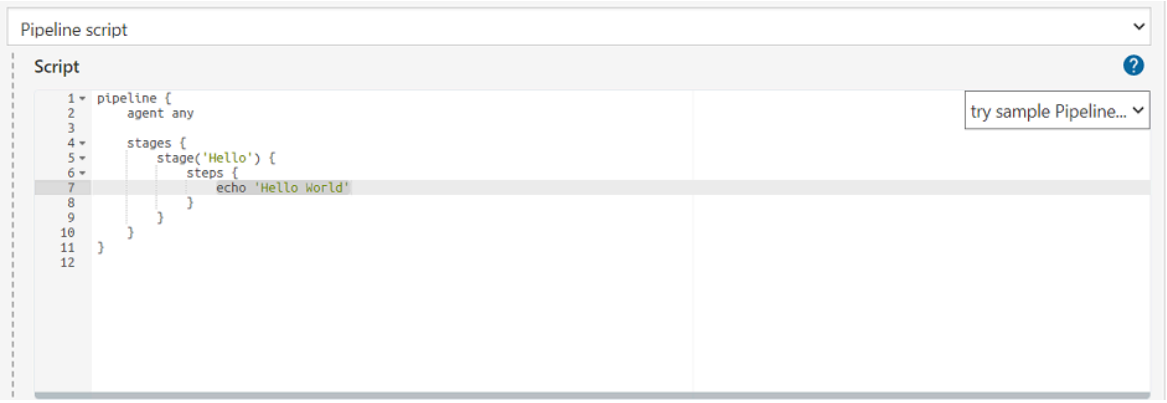
Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 504.
- Created a pipeline script. See Creating a pipeline script from Jenkins on page 507.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.
- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Select the **Pipeline script** option in the **Definition** field.
4. Select the sample script from the drop-down list.

For example, if you select `Hello Word` from the drop-down list, then the sample script is displayed as follows:



5. Perform the following steps to edit the sample script:

    a. Replace the agent `any` in the script with the name of the label that you provided when you created the Agent.
    For example, if you provided a name of the label as `win10_ft_1022`, then edit the script as follows:

    ```
    agent {label 'win10_ft_1022'}
    ```

    b. Provide a name for the `stage` in the script.
    For example, `stage('Run FTUI Demo')`

    c. Specify the script that you created in step 7 on page 508 in place of *"`echo 'Hello Word'`"*.

6. Click **Save**.

Results

You have configured the **Pipeline** project by using the **Pipeline script** option.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 510.

Configuring the Pipeline project by using the script from SCM

You can configure the **Pipeline** project to use the pipeline script from the Source Code Management (SCM), and then run test assets from the Jenkins **Pipeline** project.

Before you begin

You must have completed the following tasks:

- Installed the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test on the Jenkins primary server. See Installing the plugin on the Jenkins primary server on page 504.
- Created an Agent in Jenkins. For more information about creating Agents, refer to the Jenkins documentation.

- Copied the name of the labels that you provided in the **Labels** field when you created the Agent.
- Created a pipeline script and committed the script to the Git repository. For more information, refer to the Jenkins documentation.
- Been granted administrator permission of the Jenkins server to add credentials into Jenkins.
- Added your GitHub credentials into Jenkins. For more information about adding global credentials, refer to the Jenkins documentation.
- Created a Jenkins **Pipeline** project.

About this task

If your pipeline scripts are complex, you can then write the script outside of Jenkins UI. You must commit that script file into your Git repository. During the **Pipeline** build process, Jenkins checks out the script file from the Git repository and runs your **Pipeline** project.

1. Open your Jenkins **Pipeline** project from the list.
2. Click **Configure**, and then select the **Pipeline** tab.
3. Select the **Pipeline script from SCM** option in the **Definition** field.
4. Select the **Git** option from the drop-down list in the **SCM** field.
5. Enter the URL of the Git repository in the **Repository URL** field where you stored your pipeline script.
6. Select the credentials of the Git repository from the drop-down list.
7. **Optional:** Enter the branch name of the Git repository in the **Branch Specifier** field.
8. Enter the path of the script that you stored in the Git repository in the **Script path** field.
9. Click **Save**.

Results

You have configured the **Pipeline** project by using the **Pipeline script from the SCM** option.

What to do next

You can run test assets from the Jenkins server. See Running tests from Jenkins on page 510.

Running tests from Jenkins

You can run test assets either from the Jenkins **Freestyle** or the **Pipeline** project on the Jenkins server to test an application under test.

Before you begin

You must have completed the following tasks:

- Verified that you have test assets residing within Rational® Functional TesterHCL OneTest™ UI.
- Configured the **Freestyle** project, if you want to run test assets from the **Freestyle** project. See Configuring the Freestyle project on page 505.

- Configured the **Pipeline** project either by using the **Pipeline script** or **Pipeline script from SCM**, if you want to run test assets from the **Pipeline** project. See Configuring the Pipeline project by using the sample script on page 508 or Configuring the Pipeline project by using the script from SCM on page 509.
- Downloaded the latest **Test Results Analyzer** plugin from the Jenkinsci repository. For more information about the **Test Results Analyzer** plugin, refer to the Jenkins documentation.
- Installed the **Test Results Analyzer** plugin on the Jenkins server. For more information about installing the plugin, refer to the Jenkins documentation.

About this task

After the test run is complete, you can view the test results in a tabular format when you use the Jenkins **Test Results Analyzer** plugin. With a TRA-based report, you can filter the results based on the status of the build. The status of the builds can be PASSED, FAILED, and SKIPPED. You can also identify the failed build from the TRA-based report and take the required action to resolve the build errors. The TRA report also represents the build information in **Line**, **Bar**, and **Pie** graphs so that you can easily read the data.

Note: You can view the TRA-based reports only when you run Web UI tests.

1. Log in to the Jenkins server.
   Result
   The Jenkins dashboard is displayed.
2. Open your Jenkins **Freestyle** or **Pipeline** project from the list.
3. Perform one of the following steps to run the test assets from Jenkins:
   - Click **Build Now**.
   - Click **Build with Parameters** to provide the values for the defined parameters, and then click **Build**.

Results

You have run the test from the Jenkins server.

What to do next

You can view the build logs by clicking the build number from the **Build History** pane, and then selecting the **Console Output** option.

You can also view the history of test results in a tabular format by clicking the **Test Results Analyzer** option. The TRA reports are available only for Web UI test assets.

Testing with Maven

Starting with v9.2, you can use the Maven plug-in to run Functional Test scripts. Apache Maven is a software build tool based on the concept of a project object model (POM). You can use the Maven plug-in that is provided to run tests as part of your Maven build.

Note: This is an experimental feature in 9.2.

Introduction

To automate testing with Maven, you must configure a POM file that is delivered with the product installation package and launch the test from command line using Maven command.

Before you begin

- You must have installed Rational® Functional TesterHCL OneTest™ UI and set up an environment variable that points to the installation directory.
- You must have installed Maven and set up an environment variable that points to the `M2_HOME` directory.
- You must have a functional test.

Downloading FT-Maven plug-in

Maven plug-in is included in the product installation. The plug-in contains a `POM.XML file`. The first time you execute this file, Maven will automatically download the plug-in from a local repository. You can use this file to execute your tests, or use and rename the `ft_pomSample.XML file` sample file that is delivered in the product package.

Configuring the sample POM file

A sample file `ft_pomSample.XML file` is delivered with the product installation. It is saved in `/SDP/maven2` folder. The file contains all types of dependencies as well as arguments required to execute the script. You just need to copy this file in your directory and modify below arguments, datasource and other optional values.

Note:

DataStore and script name arguments are required.

Arguments `ftArgs` and `scriptArgs` are optional, they must be used only if it is required by the script.

Arguments mentioned in the `POM.XML file`:

```
<arguments>
<argument><!-- Specify DataStore path (Mandatory) --></argument>
<argument><!-- Specify script name to be executed (Mandatory) --></argument>
<argument><!-- Provide ftArgs if any (Optional) --></argument>
<argument><!-- Provide scriptArgs if any (Optional) --></argument>
</arguments>
```

In the following example, the script name is `Script1` and datasource (project) is `drive:/MyName/Project1`, so the modified arguments are:

```
<arguments>
<argument>drive:/MyName/Project1</argument>
<argument>Script1</argument>
```

```
<argument><!-- Provide ftArgs if any (Optional) --></argument>
<argument><!-- Provide scriptArgs if any (Optional) --></argument>
</arguments>
```

Running the tests

- Maven automatically identifies the POM file. So if the .xml file name is `POM.XML`, then execute the tests with the mvn test command, otherwise use the appropriate command as shown in the following table:
- **Table 3. Command line to run the tests:**

Command	Description
mvn test	If file name is `POM.xml` and you are in same directory.
mvn test -f filename.xml	If file name is other than `POM.xml` and you are in the same directory
mvn test -f FULL_PATH\fileName.xml	If you are in a different directory and `POM.xml` or use a .xml file with other name in a different directory, provide the full path.

When the tests are executed, Maven will display any type of exception if it occurs while executing the tests. The errors are related to the following scenarios:

- File name you have provided.
- Required parameters missing.
- Any wrong argument.
- Any exception occurred while executing scripts.

FT-Maven plug-in will just indicate whether the test was executed successfully or not. Reports and logs are not showing up once the test execution is completed. To see the reports, you need to check the log folder in Rational® Functional TesterHCL OneTest™ UI log folder. The log will show in the explorer mode if it is disabled in FT-maven plug-in.

Use cases

Script including script name only, with no option.

```
<arguments>
<argument>drive:/MyName/Project1</argument>
<argument>Script1</argument>
</arguments>
```

Script including argument as option.

```
<arguments>
<argument>drive:/MyName/Project1</argument>
<argument>Script1</argument>
<argument>scriptArgs1 scriptArgs2 scriptArgsN</argument>
```

```
</arguments>
```

Script including `ftArgs` related options only. You can add whatever options you need here.

```
<arguments>
<argument>drive:/MyName/Project1</argument>
<argument>Script1</argument>
<argument>-rt.log_format html -log testLogFolder -iterationCount 10</argument>
</arguments>
```

Script including all options.

```
<arguments>
<argument>drive:/MyName/Project1</argument>
<argument>Script1</argument>
<argument>-rt.log_format html -log testLogFolder -iterationCount 10</argument>
<argument> scriptArgs1 scriptArgs2 scriptArgsN</argument>
</arguments>
```

Testing with IBM® Engineering Workflow Management

IBM® Rational® Functional TesterHCL OneTest™ UI can be integrated with IBM® Engineering Workflow Management (formerly known as Rational Team Concert). You can manage functional test assets by using the Jazz™ source control management that the Jazz Eclipse Client provides. You must have a compatible version of IBM® Engineering Workflow Management server set up to use the Jazz source control management.

You can use IBM® Engineering Workflow Management to connect to compatible Jazz servers, including IBM® Engineering Workflow Management servers.

Use this feature to do these tasks:

- Access IBM® Engineering Workflow Management.
- Manage functional test assets by using Jazz source control management.

The client version and server version of IBM® Engineering Workflow Management must match. See the IBM® Engineering Workflow Management information center to learn more about server and client version compatibility. To access Jazz work items, use the Work Items view in the Work Items perspective. To switch to the Work Items perspective, click **Window > Open Perspective > Work Items**.

To manage functional test assets using IBM® Engineering Workflow Management, you must install IBM® Engineering Workflow Management in the same package group as Rational® Functional TesterHCL OneTest™ UI.

Engineering Workflow Management

You can use IBM® Engineering Workflow Management as your software configuration management system to maintain functional test assets.

Using IBM® Engineering Workflow Management, you can share projects, scripts, script templates, test datasets, and object maps across the testing team. You can manage changes in test assets in a collaborative manner to track the changes efficiently.

Note: The integration of IBM Rational® Functional TesterHCL OneTest™ UI with IBM® Engineering Workflow Management is not supported in .NET.

IBM® Engineering Workflow Management must be purchased separately. You must install IBM® Engineering Workflow Management using Installation Manager and then install IBM® Engineering Workflow Management using the same package group with Rational® Functional TesterHCL OneTest™ UI.

With IBM® Engineering Workflow Management, you can manage test assets and perform source control operations such as these:

- Check-in files from your local workspace to your repository workspace.
- Make changes to the contents of your local workspace and check-in the files so that the local workspace and repository workspace contain the same versions of the files. In the repository workspace, related changes are collected as change sets so that changes in multiple files and folders can be committed in a single operation.
- Deliver the change sets to a stream so that the content in the repository workspace is made available to other team members.

Note: When you perform source control operations in the **Functional Test Projects**, the results might be inconsistent. Perform all source control operations such as check-in and deliver from the **Pending Changes** view.

For more information on Jazz™ source control, see the IBM® Engineering Workflow Management information center.

Switching to Jazz source control

You can use Jazz source control management to manage functional test assets such as test object maps and scripts to facilitate functional testing efforts. The integration is controlled by a property that is set in the `ivory.properties` file in the Rational® Functional TesterHCL OneTest™ UI installation directory. You can use one software configuration management client type only in a session.

About this task

To switch to Jazz source control, you must specify the software configuration management client type in the ivory.properties file. To modify the `ivory.properties` file, you must have the appropriate permission.

1. Close Rational® Functional TesterHCL OneTest™ UI.
2. Open the `ivory.properties` file available in the folder `<product installation directory>\Functional Tester\bin\`.
3. Set the `rational.test.ft.cm.clienttype` to `CCRC`. The default value is set to `NATIVE_CC`. Changing the value to `CCRC` enables Rational® Functional TesterHCL OneTest™ UI to use source control plug-ins like Jazz source control and CCRC Eclipse plug-ins.
4. Save the `ivory.properties` file, and start Rational® Functional TesterHCL OneTest™ UI.

Sharing a project

To manage the Rational® Functional TesterHCL OneTest™ UI projects by using the Jazz source control provider, you must share the project and select `Jazz source control` as the repository provider. To perform source control operations on the Rational® Functional TesterHCL OneTest™ UI project, the project must be in the Jazz repository workspace. When the project is shared, the project is moved to the Jazz repository workspace.

1. Right-click the Rational® Functional TesterHCL OneTest™ UI project, and click **Team > Share project**.
2. Select **Jazz Source Control** on the Share project page, and click **Next**.
3. Specify the repository workspace component to which to move this project.

    a. Choose **Select a component in an existing repository workspace** if you have an existing repository workspace.
        - To place the project in an existing component, expand the workspace, and select the component.
        - To place the project in a new component in an existing repository workspace, select the workspace, click **New Component**, enter a name for the new component, click **OK**, and select the new component in the list.

    b. Choose **Create a new repository workspace named** if you have not created a repository workspace or if you want to create a new one, type a name for the workspace, and click **Next**. Select **Share with a component from an existing stream**, expand the stream, and select a component.

    Note: If the IBM® Engineering Workflow Management server is renamed, make sure that you update the hosts file with the new server name. You must also update the repository connection with the new server name so that you can continue performing source control operations on the functional test project. If you are using a version of the IBM® Engineering Workflow Management client that is older than 4.0, restart the client after you update the new server name in the hosts file and the repository connection.

4. Click **Next**.
5. Click **Finish**. The selected project is moved to the repository workspace.

    Result

    The project is added to a component in a repository workspace. You can perform source control operations on the project. For more information on Jazz source control, see the IBM® Engineering Workflow Management information center.

Related information

ShowMe

Merging object maps

You can merge two object maps by comparing your local copy with the repository copy of the file. Rational® Functional TesterHCL OneTest™ UI merges the two object maps automatically. In cases where the object maps cannot be merged automatically, you can manually merge the object maps. The user interface displays the two object maps to be merged. The left pane displays the local copy and the right pane displays the server copy. The difference between the two object maps is highlighted. You can move the contents from right to left to merge the changes.

1. In the Functional Test projects view, right-click the project and synchronize the changes.
   Result
   The incoming and outgoing changes are displayed in the Pending Changes view. The object map with conflicting changes is preceded by a red icon.
2. Select the object map with conflicting changes. You can view the differences between the two files by comparing them. The changes are highlighted in the Object Map Compare editor.
3. Merge the changes from the delivered copy with the local copy. You can take these actions:
   Choose from:
   - **Merge the changes individually:** Select the change to be merged and click the **Copy Current Change from Right to Left** button. You can move to the next change using the **Next Difference** button or previous change using the **Previous Difference** button and merge the changes.
   - **Merge all the changes:** Click the **Copy All from Right to Left** button to merge all the changes.
4. Click **Save** to save your changes.
5. Deliver the merged object map.

Testing with Tivoli Composite Application Manager

Rational® Functional TesterHCL OneTest™ UI can be integrated with IBM Tivoli® Composite Application Manager. You can schedule the interval at which the application manager agent runs to track the response time and generate logs using Rational® Functional TesterHCL OneTest™ UI scripts.

Before you begin

Before you start using the Rational® Functional TesterHCL OneTest™ UI integration feature with IBM Tivoli Composite Application Manager, verify that the following prerequisites are met:

Database	DB2®
Features and sub components	• IBM Tivoli Monitoring Tivoli Enterprise Management Server - part (CR62TML) • ITM Portal Server for visualization - part (CR62TML) • ITCAM Application Management Console - part (CR7X4ML) • ITCAM Robotic Response Time Agent - part (CR7X4ML)
Products versions	The supported version of Tivoli Composite Application Manager, see Support Matrix.

1. Install Rational® Functional TesterHCL OneTest™ UI.
2. Install Tivoli Composite Application Manager plug-ins into the Eclipse shell so that Rational® Functional TesterHCL OneTest™ UI is configured to work with application manager.
3. Click **File > Export** and select **ITCAM** under Other section to export Rational® Functional TesterHCL OneTest™ UI scripts to Tivoli Composite Application Manager.
4. Schedule the script execution from Tivoli Composite Application Manager.
   At the scheduled interval, application manager agent runs the scripts and transaction logs gets generated.

Testing with HCL® Launch

You can integrate Rational® Functional TesterHCL OneTest™ UI with HCL® Launch to run tests as a **Process** from HCL® Launch.

When you use HCL® Launch for automating the deployments of your applications to an environment during the development cycle, you can create tests for your application in Rational® Functional TesterHCL OneTest™ UI. You can then run those tests from HCL® Launch.

You must use the HCL® Launch plugin to integrate Rational® Functional TesterHCL OneTest™ UI with HCL® Launch. Integrating Rational® Functional TesterHCL OneTest™ UI with HCL® Launch automates the process of test runs. If you have many tests to run at regular intervals, you can use HCL® Launch to initiate test runs automatically.

The following table lists the tasks that you must perform to run tests from HCL® Launch:

Tasks	Go to...
Create functional tests in Rational® Functional Tester-HCL OneTest™ UI to test your application.	Test Author Guide on page 532.
Install HCL® Launch with at least one agent.	HCL® Launch documentation. System requirements on page 28 for information about specific versions of HCL® Launch requirements.
Install the Rational® Functional TesterHCL OneTest™ UI Launch plugin on HCL® Launch.	Installing the plugin on HCL Launch on page 519.
Create a **Component** in HCL® Launch.	Creating a component in HCL Launch on page 519.
Create a **Process** for the component in HCL® Launch.	Creating a process in HCL Launch on page 520.
Configure the **Process** that you created for the component in HCL® Launch.	Configuring the process on page 521.
Create a **Resource** in HCL® Launch.	Creating a resource in HCL Launch on page 523.
Configure the **Resource** in HCL® Launch.	Configuring the resource on page 524.
Create an **Application** in HCL® Launch.	Creating an application in HCL Launch on page 525.

Tasks	Go to...
Configure the **Application** that you use to run tests in HCL® Launch.	Configuring the application on page 526.
Run the tests from HCL® Launch.	Running tests from HCL Launch on page 528.

Installing the plugin on HCL® Launch

You must install the Rational® Functional TesterHCL OneTest™ UI Launch plugin to run tests from HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Installed HCL® Launch. For more information about the installation, refer to the HCL® Launch documentation.
  - Been granted access to HCL® Launch.
  - Downloaded the Rational® Functional TesterHCL OneTest™ UI Launch plugin `HOT-UI-LAUNCH-FunctionalTest-5.2` from the HCL® License & Delivery portal.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Settings**.
3. Click **Automation Plugins** from the **Automation** pane.
4. Click **Load Plugin**.
5. Click **Choose File** to locate and open the compressed Rational® Functional TesterHCL OneTest™ UI Launch plugin file.

   Remember: You must not extract the Rational® Functional TesterHCL OneTest™ UI Launch plugin compressed file contents.

6. Click **Submit**.
   The installed plugin is displayed on the **Automation Plugins** tab.

Results

You have installed the Rational® Functional TesterHCL OneTest™ UI Launch plugin on HCL® Launch.

What to do next

You must create a **Component** in HCL® Launch. See Creating a component in HCL Launch on page 519.

Creating a component in HCL® Launch

You must create a component to include artifacts and processes. The artifacts include runnable files, images, databases, configuration instructions. Whereas the processes define the activities that components can perform.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**, and then click **Create Component**.
3. Enter a name for the component in the **Name** field.
4. Enter the details in the other optional fields based on your requirement, and then click **Save**.
   The component that you created is displayed.

Results

You have created the component in HCL® Launch.

What to do next

You must create a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 520.

Related information

HCL Launch Documentation

Creating a process in HCL® Launch

You must create a process for the component to include step properties for the test that you want to run from HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 519.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**.
   A list of components that are available in HCL® Launch is displayed.
3. Select the component from the list for which you want to create a process.
4. Click the **Processes** tab, and then click **Create Process**.
   The **Create Process** dialog is displayed.
5. Enter a name for the process in the **Name** field.
6. Select **Operational (No Version Needed)** from **Process Type** drop-down list.

7. Verify the **Default Working Directory** field.

   The **Default Working Directory** field defines the location that the agent uses to run the process. The default value is `${p:resource/work.dir}/${p:component.name}`.

   Where `${p:resource/work.dir}` is the default working directory for the agent and `${p:component.name}` is the name of the component.

8. Click **Save**.
   The process that you created is listed in the **Processes** tab and the **Design** tab for the process is displayed.

Results

You have created the process for the component in HCL® Launch.

What to do next

You must configure the process in HCL® Launch. See Configuring the process on page 521.

Related information

HCL Launch Documentation

Configuring the process

You must configure the process that you created for the component to organize the steps in the process, specify the properties of the steps, and connect them.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 519.
  - Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 520.

About this task

When you open any process to configure, the process is displayed in the process editor. The process editor lists the plugins and steps. The required **Start** and **Finish** steps represent the beginning and the end of the process and are automatically placed on the design area.

You must provide the values for certain fields in the properties for the selected test step to run tests from HCL® Launch. The minimum fields that you need to configure are **Name**, **Workspace**, **Project**, and **Test Suite Name** or **Config File**. The remaining fields are optional.

For example, if you provide a complete path of a configuration file that contains the parameters for a `test` run in the **Config File** field, then you need not provide values for the **Workspace**, **Project**, and **Test Suite Name** fields.

Note: If Rational® Functional TesterHCL OneTest™ UI is not installed in default location, then you must provide the value for the **IMShared Location** field along with other mandatory fields.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Components**.
   A list of components that are available in UrbanCode™ Deploy is displayed.
3. Select the component from the list in which you created the process.
4. Click the **Processes** tab.
   A list of processes that are available for the component is displayed.
5. Select the process from the list that you want to configure.
   The **Design** tab for the process is displayed.
6. Click **HCL OneTest Studio**, and then **HCL OneTest UI** from the left menu.
7. Drag the **Run an HCL OneTest UI test - Functional Test** step, and then drop it into the design area.

   Note: The selected test must be placed between **Start** and **Finish** steps.

8. Specify the properties for the selected test by performing the following steps:

   a. Click the **Edit** icon.
      Result
      The **Edit Properties for Run an HCL OneTest UI test - Functional Test** dialog is displayed.

   b. Specify the properties for the selected test step by referring to the following table:

Table 4. Input properties for running a functional test step

Name	Type	Description	Required
Iteration Count	String	Number of dataset iterations to be run.	No
Log Format	Enumeration ▪ xml ▪ html ▪ text ▪ TPTP	Format of script execution logs.	No
Project Directory	String	Fully qualified path to the Functional Test project directory.	Yes

Name	Type	Description	Required
Script Name	String	Name of the script to be executed.	Yes
User Arguments	String	Additional user arguments for playback.	No

Note: The following fields are from HCL® Launch:

- **Working Directory**
- **Precondition**
- **Post Processing Script**
- **Use Impersonation**
- **Auth Token Restriction**

You can accept the default values or change the values based on your requirements. For more information about these fields, see the related links.

c. Click **OK** to save the properties for the test.

9. Click **Save** in the design area.

Results

You have configured the process for the component in HCL® Launch.

What to do next

You must create a resource in HCL® Launch. See Creating a resource in HCL Launch on page 523.

Related information

Creating a command-line config file on page 1400

Viewing test logs on page 1493

Generating offline user tokens

HCL Launch Documentation

Process step preconditions

Post-processing scripts

User impersonation for process steps

Restricting authentication tokens

Creating a resource in HCL® Launch

You must create a resource to associate agents with components that you created in HCL® Launch.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Resources**, and then click **Create Top-Level Group**.
   The **Create Resource** dialog is displayed.
3. Enter a name for the resource in the **Name** field.
4. Click **Save**.

Results

You have created the resource in HCL® Launch.

What to do next

You must configure the resource. See Configuring the resource on page 524.

Related information

HCL Launch Documentation

Configuring the resource

You must configure the resource to add an agent and associate the agent with the component.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 519.
  - Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 523

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Resources**.
   Result
   A list of resources that are available in HCL® Launch is displayed.
3. Perform the following steps to add an agent to the resource:
   a. Click the resource from the list for which you want to add an agent.
   b. Click the **Actions** icon from the last column, and then click **Add Agent**.
   c. Select the agent from the drop-down list.

Note: The **Name** field is auto populated with the name of the agent.

d. Click **Save**.

Result

The selected agent is added to the resource and you can view the status of the agent in the **Status** column.

4. Perform the following steps to add a component to the agent:

a. Click the agent from the list for which you want to add a component.

b. Click the **Actions** icon from the last column, and then click **Add Component**.

c. Select the component from the drop-down list.

Note: The **Name** field is auto populated with the name of the component.

d. Click **Save**.

Result

The selected component is added to the agent.

Results

You have configured the resource in HCL® Launch.

What to do next

You must create an application. See Creating an application in HCL Launch on page 525.

Related information

HCL Launch Documentation

Creating an application in HCL® Launch

You must create an application to fetch all the components together that you want to deploy.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have been granted access to HCL® Launch.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
3. Click **Create Applications**, and then **New Applications**.

4. Enter a name for the application in the **Name** field.
   The **Environments** page for the application that you created is displayed.

Results

You have created the application in HCL® Launch.

What to do next

You must configure the application. See Configuring the application on page 526.

Related information

HCL Launch Documentation

Configuring the application

You must configure the application to associate resources with environments and define processes to run tests.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 519.
  - Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 520.
  - Configure the process for the component in HCL® Launch. See Configuring the process on page 521.
  - Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 523.
  - Configured the resource in HCL® Launch. See Configuring the resource on page 524.
  - Created an application in HCL® Launch. See Creating an application in HCL Launch on page 525.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
   Result
   A list of applications that are available in HCL® Launch is displayed.
3. Click the application that you want to configure from the **Name** column.
   The **Environments** page for the selected application is displayed.
4. Perform the following steps to create an environment for the application that you selected:
   a. Click **Create Environment**.
   b. Enter a name for the environment in the **Name** field.
   c. Click **Save**.
5. Perform the following steps to configure resources to the environment:

a. Click the environment that you created.

b. Click **Add Base Resources**.

Result

A list of resources that are available in HCL® Launch is displayed.

c. Select the checkbox to add resources to the environment.

d. Click **Save**.

Result

You can view the corresponding agent and the component that you added for the resource by using the **Expand** icon.

6. Perform the following steps to add the component to the application:
   a. Click **Applications**, and then select your application from the list.
   b. Click the **Components** tab, and then **Add Components**.
   c. Select the checkbox from the drop-down list to add components to the application.
   d. Click **Save**.
7. Perform the following steps to create a process for the application:

   a. Click the **Processes** tab, and then **Create Process**.

   b. Enter a name for the process in the **Name** field.

   c. Click **Save**.

   Result

   The **Design** tab for the process that you created is displayed.

8. Drag the component process listed under the **Component Process Steps** option from the left navigation pane and drop it into the design area.
9. Select the component process from the drop-down list in the **Operational (No Version Needed) Process** field.
10. Click **Save**.
11. Click the **Edit** icon, and then change the name of the properties.
12. Click **OK**, and then click **Save**.

Results

You have configured the application to run tests from HCL® Launch.

What to do next

You can run tests from HCL® Launch. See Running tests from HCL Launch on page 528.

Related information

HCL Launch Documentation

Running tests from HCL® Launch

You can run the functional tests for your application under test from HCL® Launch as a **Process**.

Before you begin

- You must be familiar with working with HCL® Launch.
- You must have performed the following tasks:
  - Been granted access to HCL® Launch.
  - Created a component in HCL® Launch. See Creating a component in HCL Launch on page 519.
  - Created a process for the component in HCL® Launch. See Creating a process in HCL Launch on page 520.
  - Configure the process for the component in HCL® Launch. See Configuring the process on page 521 .
  - Created a resource in HCL® Launch. See .Creating a resource in HCL Launch on page 523
  - Configured the resource in HCL® Launch. See Configuring the resource on page 524.
  - Created an application in HCL® Launch. See Creating an application in HCL Launch on page 525.
  - Started the HCL® Launch agent. For more information, refer to the HCL Launch documentation.
  - Closed Rational® Functional TesterHCL OneTest™ UI.

1. Log in to HCL® Launch, if you are not already logged in.
   The HCL® Launch dashboard is displayed.
2. Click **Applications**.
   Result
   A list of applications that are available in HCL® Launch is displayed.
3. Click the application that you configured for the test run.
4. Click the **Request Process** icon.
   Result
   The **Create Deployment** page is displayed.
5. Select the process that you created for the application which contains the test assets from the **Process** drop-down list.
6. Click **Next**, and then select the specific version of component, if any.

   Note: If you have multiple versions for your component, then only you can choose the required version by clicking the **Add** option.

7. Click **Next**, and then verify the properties for the test run.
8. Click **Next**, and then verify that the status of the **Run Now** option is enabled.
   You can disable the **Run Now** option, and then use the **Date**, **Time**, and **Recurrence Pattern** fields to run tests as **Process** at the scheduled time.
9. Click **Next**, and then verify the details about the **Process**.
10. Click **Submit Deployment** to initiate the test run.

Results

You have run the functional test as **Process** from HCL® Launch.

What to do next

You can view the status of the process as follows:

- **Success**: When the test run is successful
- **Failed**: When the test run fails

You can view the details of the test run in the **Log** tab. The **Reports information** section on the Log file displays the names of the report along with its corresponding URLs if you publish test results to IBM® Rational® Test Automation ServerHCL OneTest™ Server.

Testing with UrbanCode™ Deploy

You can run functional test scripts remotely with the IBM® Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin for UrbanCode™ Deploy.

The UrbanCode™ Deploy plugin for Rational® Functional TesterHCL OneTest™ UI includes steps for running functional test scripts remotely on UrbanCode Deploy agent computers. Rational® Functional TesterHCL OneTest™ UI integration with UrbanCode Deploy is supported for Java Scripting and Simplified Scripting in the Eclipse IDE. Test scripts recorded in the Visual Studio IDE are not supported.

Compatibility

This plugin requires UrbanCode™ Deploy version 6.1 or later and Rational® Functional TesterHCL OneTest™ UI version 8.6.0.3 version 9.1 or later.

Installing plugins in UrbanCode™ Deploy

1. Download the installation files for the Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin for UrbanCode™ Deploy from the HCL® License & Delivery portalIBM WebSphere, Liberty & DevOps Community.
2. Load the plugin in UrbanCode™ Deploy.
    a. For automation plugins, click **Settings> Automation Plugins**, then click **Load Plugin**.
    b. For source plugins, click **Settings> Source Config Plugins**, then click **Load Plugin**.
    c. Enter the path to the compressed plugin file, and then click **Submit**.

The plugin is listed on either the **Automation Plugins** pane or the **Source Config Plugins** pane. After the plugin is installed, it is available immediately.

Input properties for running a functional test step

Run a test using Rational® Functional TesterHCL OneTest™ UI.

Table 5. Input properties for running a functional test step

Name	Type	Description	Required
Iteration Count	String	Number of dataset iterations to be run.	No
Log Format	Enumeration • xml • html • text • TPTP	Format of script execution logs.	No
Project Directory	String	Fully qualified path to the Functional Test project directory.	Yes
Script Name	String	Name of the script to be executed.	Yes
User Arguments	String	Additional user arguments for playback.	No

Running functional tests

Use this plugin to continuously initiate the launch of functional tests from UrbanCode™ Deploy. Before you begin:

- Install the UrbanCode Deploy agent and connect it to the UrbanCode Deploy server. For details, see the UrbanCode Deploy documentation.
- Install Rational® Functional TesterHCL OneTest™ UI on the UrbanCode Deploy agent computer and apply a valid license.
- Enable the functional testing environment (for example, Java) on the UrbanCode Deploy agent. If you intend to test a web-based application, enable any browser supported by Rational® Functional TesterHCL OneTest™ UI, that is, Microsoft Internet Explorer, Mozilla Firefox, or Google Chrome. For information about enabling the functional testing environment, see Preparing the functional test environment on page 774.
- Configure the application to be tested (AUT) on the UrbanCode Deploy agent. For details, see Configuring applications for testing on page 796.
- If you have stored the functional test scripts within a folder, the value of the **Script Name** input property should be **folder name.script name**, for example, `Mytests.app1test`.

As a tester, you might have a large number of regression tests to be run against the latest builds of a product. Instead of manually running the tests against every new build, you can install the latest build on an UrbanCode™ Deploy Agent computer and let it automatically start the tests for you.

After deploying the Rational® Functional TesterHCL OneTest™ UI - Functional Test plugin on the UrbanCode Deploy server, create a component and its processes, applications and its processes, environments, and resources. For information about how to create and configure these pieces, see https://www.ibm.com/support/knowledgecenter/SS4GSP/ucd_welcome.html.

After deployment, the UrbanCode Deploy plugin for Rational® Functional TesterHCL OneTest™ UI is available on the UrbanCode Deploy server. The plugin for Rational® Functional TesterHCL OneTest™ UI includes only a single step: Run Functional Tests. The plugin for Rational® Functional TesterHCL OneTest™ UI includes only a single step: Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI test - Functional Test.The component process runs a functional script that is part of a Rational® Functional TesterHCL OneTest™ UI project previously stored on an UrbanCode Deploy agent.

To create a workflow, do the following tasks:

1. Click the component process. The Tools view displays the available plugin steps.
2. From IBM Rational Functional Tester - Functional TestHCL OneTest UI - Functional Test, drag the **Run an IBM Rational Functional Tester - Functional TestRun an HCL OneTest UI test - Functional Test** step into the design space.
3. Specify the properties for the step. For information about the properties, see Input properties for running a functional test step on page 529.
4. After configuring the properties, save the step.

Note:

- You must compile the functional test scripts in the Eclipse IDE before running them on UrbanCode Deploy.
- The UrbanCode Deploy output variable called **TestResult** contains the overall playback result of the test scripts. This variable helps you view the test results on UrbanCode Deploy itself rather than on the test computer where the scripts are run.
- If the execution of a Rational® Functional TesterHCL OneTest™ UI script is dependent on an external .jar file, make sure you place the .jar file in the **customization** folder. Adding references to the file under project properties (which is what you normally do when playing scripts from the Rational® Functional TesterHCL OneTest™ UI IDE) will not suffice.

Chapter 7. Test Author Guide

This guide describes how to create test scripts and enhances them by applying different test elements such as dataset, variables, and verification points.

Testing in the UI Test perspective

When you develop web applications, Windows-based applications, or mobile applications, you can use IBM® Rational® Functional TesterHCL OneTest™ UI to create functional tests for these applications. You must first record the tests and then use Rational® Functional TesterHCL OneTest™ UI to run the tests before you can view the test results.

You can find the following information:

- Testing web applications on page 532
- Testing mobile applications on page 687
- Testing Windows desktop applications on page 707
- Recording SAP tests on page 714
- Working with Selenium or Appium tests on page 733
- Compound tests on page 737
- Accelerated Functional Tests on page 745
- Working with keywords on page 770

Testing web applications

You can test web applications in Rational® Functional TesterHCL OneTest™ UI by using the industry-standard browsers such as Google Chrome, Apple Safari, Mozilla Firefox, and Microsoft Edge. You can record Web UI tests and then play back the tests to evaluate the test results. You can also test the web applications on dual monitors when you extend the display of your computer to a secondary monitor.

Creating Web UI tests

See the different ways to record Web UI tests. After the recording stops, the tests are generated automatically.

Web UI recording

IBM® Rational® Functional TesterHCL OneTest™ UI provides multiple ways to create Web UI tests using the UI Test perspective. You can create a Web UI test that captures both functional and HTTP traffic in the same recording. In addition, you can create a Web UI test for a web application that is already running in an instance of the Chrome, Firefox, or Safari browser. You can also initiate a recording from a step of an existing test. You can also play back tests on Microsoft™ Edge browser but they must be recorded on Chrome, Firefox, or Internet Explorer.

Recording a Web UI test that captures both functional and HTTP traffic in the same recording

With this style of recording, Rational® Functional TesterHCL OneTest™ UI starts the browser and configures the test environment before you start the recording. This style of recording provides a unified recording capability that lets

you capture both functional and HTTP traffic in the same recording. As a result, you can generate both a Web UI functional test and an HTTP load test from the same recording session.

Support is provided for Chrome, Firefox, and Internet Explorer on Windows™ computers, Chrome and Firefox on Linux® computers, and Chrome and Firefox on Macintosh computers.

Recording a Web UI test using a running browser instance

With this style of recording, you can record functional tests for web applications that are already running in an existing browser tab or window. You cannot, however, generate HTTP load tests. To use this style of recording, you must install a Web UI browser extension for each supported browser.

Support is provided for Chrome and Firefox on Windows™ and Linux® computers and for Safari, Chrome, and Firefox on Macintosh computers. Internet Explorer is not supported.

Recording Web UI steps to add to an existing test

If you already have a test and want to add more steps to it, open the test and initiate the recording from a step after which you want to add the new steps. This style of recording is useful when the task flow of an application has changed in a newer version of the application and you want to update the existing test script.

Recording a Web UI test in the private mode of a browser

While you record Web UI tests, if you do not want the user profile that is stored in the browser to affect your test recording, then you can use the private or incognito mode. Also, when you do not want cookies and caches of the browser to impact your test recording, you can choose the private mode.

Using browser arguments during the recording a Web UI test

Before you start recording Web UI tests, you can specify the arguments to be used by the browser. Rational® Functional TesterHCL OneTest™ UI launches the browser with the specified arguments. You can specify multiple arguments separated by a semicolon.

Variable for storing the name of the browser used to record the test

Prior to 9.1.1, you could define a variable in a test to specify the web browser to use for running the test. The feature was available for running a test from the command line, as part of a schedule from IBM® Rational® Performance TesterHCL OneTest™ Performance, or from IBM® Rational® Quality Manager. The reserved name for this variable is `RTW_WebUI_Browser_Selection`. However, after defining the variable, if you ran the test from the Web UI Test perspective, the browser selected in the **Run configuration** dialog box took precedence over the browser specified in the variable.

Now, IBM® Rational® Functional TesterHCL OneTest™ UI creates a test variable automatically whenever a new Web UI test is recorded or when an old test is used that does not already have this variable defined.

You can use this test variable in **If** conditions to assign different behavior for different browsers. Doing so allows you to create more robust tests that will run successfully in more than one browser.

During test execution, the value of the test variable is set to the name of the browser on which the test is being run. If you select Firefox in the run wizard, the value of the variable is set to Firefox, thus ignoring the original value that was set in the test during recording or while editing the test. For a command line or a Schedule execution, the value that was set in the test is used, since in these cases there is no run wizard.

During test execution, an **If** condition accepts the following value names:

- Firefox
- Chrome
- Internet Explorer
- Safari
- Microsoft™ Edge

Related information

Defining a variable to run a test with a selected browser on page 578

Adding conditional logic to tests on page 589

Prerequisites for creating tests

Before you can create a test, you must complete the prerequisite tasks.

When you use IBM® Rational® Functional TesterHCL OneTest™ UI to record a test for the application under test (AUT), you must ensure that you follow the prerequisite conditions mentioned or complete the prerequisites tasks.

Find information about the conditions that are independent of the web browsers that you must complete:

- You must wait for each page to load completely when you record a test.

  Note: This waiting time does not affect performance results because you can remove extra waiting time (think time) when you configure a run to play back the test.

- You must enable the JavaScript option in a web browser to record and play back Web UI scripts.
- You must install the Mozilla Firefox, Microsoft Edge, or Google Chrome browsers on computers that run Mac operating system at the default location of `/Applications` when you want to record or play back a Web UI test.

Prerequisites for using the Google Chrome browser

You must complete the following tasks when you want to use the Google Chrome browser for recording web applications:

- Install the Google Chrome extension for Web UI testing when you want to record web applications that run on the Google Chrome browser. See Enabling Google Chrome for Web UI testing on page 540.
- Enable the Google Chrome Device Mode feature to emulate web applications on mobile devices when you want to record such tests. See Recording a test with Google Chrome Device Mode on page 568.

Prerequisites for using the Microsoft Edge browser

You must complete the following task when you want to use the Microsoft Edge browser for recording web applications:

- Install the Edge extension for Web UI testing when you want to record web applications that run on the Edge browser. See Enabling Microsoft Edge for Web UI testing on page 542.

Prerequisites for using the Internet Explorer browser

You must complete the following tasks when you want to use the Internet Explorer browser for recording web applications:

- Add `<!DOCTYPE html>` at the beginning of the HTML source of the pages to ensure that the browser always loads the web pages in standard mode.
- Click **Tools > Compatibility View Settings** and clear all the selections to prevent the application from running on a compatibility mode internally.
- Enable the Rational® Functional TesterHCL OneTest™ UI extension in the browser when you are recording the test with Internet Explorer for the first time by performing any of the following actions:
    - Click **Enable** in the dialog that appears the first time you start Internet Explorer after you installed Rational® Functional TesterHCL OneTest™ UI.
    - Open Internet Explorer and click **Tools > Manage add-ons**. Then, select **RtwIEBhoWithJS Class**, and then click the **Enable** option.

Prerequisites for using the Mozilla Firefox browser

You must complete the following tasks when you want to use the Mozilla Firefox browser for recording web applications:

- Install the Web UI browser extension for Firefox®. See Enabling Mozilla Firefox for Web UI testing on page 543.
- Check the **Use an alternate Firefox profile** option in the **Recorder Settings dialog**, and select a user profile that is associated with the extension.
- Clear the Firefox® cache before you start the Firefox® browser so that the browser extension can run and record the web applications.

Prerequisites for using the Safari browser

You must complete the following tasks when you want to use the Safari browser for recording web applications:

- Enable the Safari browser when you want to record Web UI tests. See Enabling the Apple Safari browser to perform Web UI tests on macOS on page 538.
- Click the **Develop > Allow Remote Automation** to be able to record a test.
- Click **Stop Session** in the dialog that is presented when you start the recording of the web application in the Safari browser.

Configuring applications for tests

When you want to test Android, iOS, web, or Windows desktop applications, you must first configure them. You can configure these applications in a common web interface that helps you to manage all the applications in one place. You can use these applications anytime later to record and play back tests.

1. Go to the **Web UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI.
2. Click the **Application Configuration** icon  in the toolbar.

   A browser window opens and the **Application Configuration** page is displayed. The page displays the list of all applications that you have configured in Rational® Functional TesterHCL OneTest™ UI. You can use the filters to view the applications based on the type.
3. Click **Add** and select the type of application that you want to configure.

   A dialog box is displayed and fields differ based on the application type.

   The application types are as follows:
   - **Android**: To configure the Android applications that you want to use for mobile tests.
   - **Desktop**: To configure the Windows desktop applications that you want to use for Windows test.
   - **iOS**: To configure the iOS applications that you want to use for mobile tests.
   - **Web**: To configure the web applications that you want to use for Web UI tests.
4. Enter the details of the application, which you want to configure, in the appropriate fields that are displayed in the dialog box.

   For example, if you want to configure a web application, then you must enter the URL of the application.

   Note: You can choose a method to provide the application details for Android and iOS applications. Select one of the following methods:
   - Select **Manually** if you want to manually enter the details in each of the fields.
   - Select **From APK or From IPA** if you want to fill in the application details automatically. You can do one of the following tasks to automatically fill in the details:
     - Browse and select the `.apk` or `.ipa` file from your local drive.
     - Drag and drop the `.apk` or `.ipa` file from your local drive to the **Application Configuration** page.
5. Click **Add**.

Results

The application is configured and it is listed on the **Application Configuration** page.

What to do next

You can perform the following actions on the configured applications:

- Edit the details of the application by clicking the **Edit** icon  inline with the name of the application.
- Delete the application if it is no longer required by clicking the **Delete** icon  .
- Apply the variable substitution, which is local to a test, to all the test suites that use the same application for their tests by clicking the **Update application in selected test suites** icon  . The changes of the URL are then applied to all the test suites that use the web application in their tests.

Synchronizing changes of the configured applications

When you modify the details of a configured application, you can synchronize these changes to the primary application or other tests that use the same application.

Before you begin

You must have configured the application in the **Application Configuration** page.

About this task

You can modify the application details either from the parent application on the **Application Configuration** page or from any test that uses the configured application. You can modify the application details and synchronize the changes in both the direction, that is, from the primary application to test suites and vice versa.

1. Access the application by performing one of the following tasks:
    - To access the primary application on the **Application Configuration** page, do the following tasks:
        a. Click the **Application configuration** icon.

        The **Application Configuration** page is displayed.

        b. Click the **Edit** icon inline with the application that you want to modify.

        The **Web application** dialog box is displayed.
    - To access the application from the test suites, do the following tasks:
        a. Click the **Launch application** step in the **Test Contents** pane.

        The **Application Details page** is displayed.

        b. Click the **Edit** icon inline with the application that you want to modify.

        The **Web application** page is displayed.
2. Make necessary changes to the configured application.
3. Save the changes by performing the following actions:

- For primary application, click **Save** in the **Web application** dialog box.
- For test suites, click **Save** to save the test suite.

4. Synchronize the application changes by selecting the other tests or primary application:
   - Perform the following actions if you modified from the primary application:
     a. Expand the application details by using the down arrow.
     b. Select one or more tests that you want to update.
     c. Click the **Synchronize** icon.
   - Perform the following actions if you modified the application details from the test suites:
     a. Click the **Synchronize** icon.

        The **Update Configuration** dialog box is displayed.

     b. Select one or more tests that you want to update.
     c. Click the **Finish** icon.

Results

You have synchronized the changes of the configured application with the test suites or the primary application.

Enabling the Apple Safari browser to perform Web UI tests on macOS

You can record and play back Web UI tests in the Apple Safari browser to test the web applications. To record Web UI tests in the Safari browser, you must first enable the browser and then record tests.

About this task

The Safari browser is ready to record Web UI tests after you enable the remote automation feature and install the Web UI extension. The Safari browser is ready to play back Web UI tests immediately after you enable the remote automation feature.

Note: The installation of the Web UI extension is required only for recording and not for playing back of Web UI tests.

1. Enable the remote automation feature by performing the following steps:
   a. Click **Safari > Preferences > Advanced** tab, and then select the **Show Develop menu in the menu bar** checkbox.

      The **Develop** menu is displayed in the menu bar.

   b. Click the **Develop** menu, and then select **Allow Remote Automation**.

      The remote automation feature is enabled.
2. Install the Web UI extension for the Safari browser by performing the following steps:

a. Navigate to the `SafariWebExtension.zip` file within the shared installation directory. For example,`<IMShared>/plugins/com.ibm.rational.test.rtw.webgui.browextension.safari_<version string>/SafariWebExtension/SafariWebExtension.zip`.
b. Double click the compressed file.

   The file is automatically extracted and saved in your system.

c. Double-click the extracted file.

   The Web UI extension is automatically installed.

   Note: If an error message is displayed, perform the following steps:
   i. Click **System Preferences > Security & Privacy > General**.

      You can view an error message in the **Allow apps downloaded from:** section.

   ii. Click the **Allow** button.

      The Web UI extension is unblocked, and you can successfully open the extension.

3. Verify whether the extension is installed, and the required browsers are enabled by performing the following steps:
   a. Open the Safari browser.
   b. Click **Safari > Preferences > Extensions**.

      The installed extension is displayed on the **Extensions** page.

   c. Select the Web UI extension.

      The details related to permissions for the selected Web UI extension are displayed on the right pane of the **Extensions** page.

   d. Perform one of the following steps:
      - Click the **Edit Websites…** button.

        You can allow the Web UI extension to read and alter specific websites that you visit by entering the list of corresponding URLs.

      - Click the **Always Allow on Every Website…** button.

        You can allow the Web UI extension to read and alter all websites.

Results

The Safari browser is enabled for recording and playing back Web UI tests.

What to do next

You can now record or play back a Web UI test in the Safari browser.

Related information

Running a Web UI test on page 1243

Enabling Google Chrome for Web UI testing

You must install the Google Chrome extension for Web UI testing, when you want to record web applications that run in the Chrome browser.

About this task

You can use the extension that is available with your Rational® Functional TesterHCL OneTest™ UI installation or from the Chrome Web Store.

1. Select from an option to install the Google Chrome extension for Web UI testing:
   - To install the extension from the installation package, go to Step 2 on page 540.
   - To install the extension from the Chrome Web Store, go to Step 3 on page 540.
2. Perform the following steps to install the use the Google Chrome extension for Web UI testing that is packaged with the product:
   a. Open the Google Chrome browser.
   b. Click **More Tools > Extensions** to open the Chrome browser extension page.
   c. Select the **Developer Mode** option.
   d. Select the **Load unpacked extension** option.
   e. Go to the `WebUIExtension` folder within the default shared directory of IBM® Rational® Functional TesterHCL OneTest™ UI.

      For example, on a Windows computer, the directory could be as follows:

      ```
      C:\Program
       Files\IBM\IBMIMShared\plugins\com.ibm.rational.test.rtw.webgui.browextension.chrome_version
      _id\ChromeExtension
      ```

      ```
      C:\Program
       Files\HCL\HCLOneTest\plugins\com.ibm.rational.test.rtw.webgui.browextension.chrome_version_i
      d\ChromeExtension
      ```

   f. Select the `WebUIExtension` folder, and then click **OK**.
3. Perform the following steps to install the use the Google Chrome extension for Web UI testing from the Chrome Web Store:

a. Open the Google Chrome browser.

b. Enter `chrome.google.com/webstore/category/extensions` in the **URL** field, and then press **Enter** or click Chrome web store.

c. Enter `Rational Functional TesterOneTest UI` in the **Search** field.

d. Press **Enter**.

The available extensions are listed.

e. Select the Rational® Functional TesterHCL OneTest™ UI - Web UI extension, and then click **Add to Chrome**.

The **Confirm New Extension** dialog is displayed.

f. Click **Add extension**.

4. Perform the following steps if you want to record tests in the private mode of a browser:

Note: If you want to record any test in the private mode, you must configure the test by selecting the private mode option for the browser in the Rational® Functional TesterHCL OneTest™ UI UI.

a. Check and ensure that you have added the Rational® Functional TesterHCL OneTest™ UI - Web UI extension to the default profile of the browser (C:\Users\<username>\AppData\Local\Google\Chrome \User Data\Default).

b. Enable only the Rational® Functional TesterHCL OneTest™ UI - Web UI extension and disable or remove the other Rational® Functional TesterHCL OneTest™ UI extension, if installed.

c. Go to **Extensions > Manage Extension** in the browser toolbar.

d. Select the following options that are listed for the Web UI extension:

- **Allow in Incognito**
- **Allow access to file URLs**

e. Close all open browsers. You can also check and end any browser processes that might be running in the background.

What to do next

Verify that the Google Chrome extension for Web UI testing is installed by performing the following actions:

- Open the Chrome browser, and then open the Chrome browser extension page by clicking **Customize and control Google Chrome > More Tools > Extensions**. You can find that the extension is installed.

Record a test using an existing instance of the Chrome browser. See Recording a Web UI test by using a running browser instance on page 565 for details.

Enabling Microsoft Edge for Web UI testing

You can record UI test for a web application by using Microsoft Edge browser in a running browser instance. You must first enable the Edge browser before you record UI tests by using the Edge browser.

About this task

For enabling the Edge browser, you must install the extension from the Chrome Web Store. Alternatively, you can also use the extension that is bundled along with Rational® Functional TesterHCL OneTest™ UI.

Note: You must preferably install the extension from the Chrome Web Store and use the extension bundled with the product only when you are unable to install from the Chrome Web Store.

1. Install the extension from the Chrome Web Store by performing the following steps:
    a. Open the Microsoft Edge browser.
    b. Click **Setting and more** icon and then select **Extensions**. The **Extensions** page is displayed.
    c. Turn on the **Allow extensions from other stores** toggle button, if it is not enabled.
    d. Click the **Chrome Web Store** link.

    The **Chrome Web Store** page is displayed.
    e. Enter `Rational Functional TesterHCL OneTest UI` in the **Search** field and press **Enter**.
    The search results are displayed.
    f. Locate **Rational Functional TesterHCL OneTest UI**, and then click **Add to Chrome**.
    A confirmation dialog is displayed.
    g. Click **Add extension**.
    The Rational® Functional TesterHCL OneTest™ UI extension is added to the Edge browser.

    Note: To verify if the extension is successfully installed on the Edge browser, click **Extensions** on the Edge browser. The added extension is listed the **Installed Extensions** section. You can enable or disable the installed extension.

2. Alternatively, use the extension that is packaged with the product.

    To install the extension:

    a. Open the Edge browser extension page by clicking **More Tools > Extensions**.
    b. Select the **Developer Mode** check box.

c. Click the **Load unpacked extension** button.

d. Navigate to the `WebUIExtension` folder within the default shared directory of IBM® Rational® Functional TesterHCL OneTest™ UI or the location where you downloaded and saved the extension.. For example, on a Windows computer, the directory could be as follows:

```
C:\Program
 Files\IBM\IBMIMShared\plugins\com.ibm.rational.test.rtw.webgui.browextension.chrome_version
_id\ChromeExtension
```

```
C:\Program
 Files\HCL\HCLOneTest\plugins\com.ibm.rational.test.rtw.webgui.browextension.chrome_version_i
d\ChromeExtension
```

e. Select the `WebUIExtension` folder and click **OK** to install the extension.

3. Perform the following steps if you want to record tests in the private mode of a browser:

Note: If you want to record any test in the private mode, you must configure the test by selecting the private mode option for the browser in the Rational® Functional TesterHCL OneTest™ UI UI.

a. Check and ensure that you have added the Rational® Functional TesterHCL OneTest™ UI - Web UI extension to the default profile of the browser (C:\Users\<username>\AppData\Local\Microsoft\Edge\User Data\Default).

b. Enable only the Rational® Functional TesterHCL OneTest™ UI - Web UI extension and disable or remove the other Rational® Functional TesterHCL OneTest™ UI extension, if installed.

c. Go to **Extensions > Manage Extension** in the browser toolbar.

d. Select the following options that are listed for the Web UI extension:

- **Allow in InPrivate**
- **Allow access to file URLs**

e. Close all open browsers. You can also check and end any browser processes that might be running in the background.

What to do next

You can record a test using an existing instance of the Edge browser. See Recording a Web UI test by using a running browser instance on page 565 for details.

Enabling Mozilla Firefox for Web UI testing

Before you can record a Web UI test of a web application that is already running in the Firefox browser, you must first install the IBM® Rational® Functional TesterHCL OneTest™ UI Web UI browser extension for Firefox.

About this task

If you are using Firefox v54 and later, you should install the new Firefox web extension. There are Web UI extensions for the different versions of Firefox: for Firefox until v53, and from Firefox v54.

To get the Web UI extension for Firefox, navigate to default shared directory `IBMIMShared` of IBM® Rational® Functional TesterHCL OneTest™ UI.

- Until Firefox v53: Two files are available `webuirecorderextension-windows.xpi` for Windows and `webuirecorderextension-linux.xpi` for Linux in the `WebUIRecorderExtension` folder.

  Note: For example:
  - On a Windows™ computer, the directory could be as follows:

    ```
    C:\Program
     Files\IBM\IBMIMShared\plugins\com.ibm.rational.test.rtw.webgui.browextension.firef
    ox_version number\WebUIRecorderExtension/
    ```

  - On a Linux™ computer, the directory could be as follows:

    ```
    /opt/IBM/IBMIMShared/plugins/com.ibm.rational.test.rtw.webgui.browextension.firefox
    _version number/WebUIRecorderExtension
    ```

- From Firefox v54 and later, the `webuirecorder_webext.xpi` file is available in the `WebUIRecorderWebExtension` folder. You can use this extension file on Windows™, on Linux™ and on Mac OS.

You can get the extension from the HCL FlexNet Operations Portal. The `webuirecorder_webext.xpi` file is available in the `Firefox` folder. You can use this extension file on Windows™, on Linux™ and on Mac OS.

1. Navigate to the appropriate folder to select the Web UI browser extension for Firefox.
2. In Firefox, click **Tools > Add-ons >** , and then click **Extensions**.
3. Drag the appropriate extension file `webuirecorderextension-windows.xpi`, `webuirecorderextension-linux.xpi` or `webuirecorder_webext.xpi` to the list of extensions on the Firefox Extensions page.
4. When prompted, select the extension, and click **Install Now**.

   Result

   Firefox displays a message indicating that the extension was installed successfully.
5. Perform the following steps if you want to record tests in the private mode of a browser:

Note: If you want to record any test in the private mode, you must configure the test by selecting the private mode option for the browser in the Rational® Functional TesterHCL OneTest™ UI UI.

a. Check and ensure that you have added the Rational® Functional TesterHCL OneTest™ UI - Web UI extension to the default-release profile of the browser (C:\Users\<username>\AppData\Local\Mozilla\Firefox\Profiles).

b. Enable only the Rational® Functional TesterHCL OneTest™ UI - Web UI extension and disable or remove the other Rational® Functional TesterHCL OneTest™ UI extension, if installed.

c. Go to **Add ons and themes > Extensions > Manage Extension** from the browser menu.

d. Enable the following option that is listed for the Web UI extension:

   - **Run in Private Windows**

e. Close all open browsers. You can also check and end any browser processes that might be running in the background.

What to do next

Record a test using an existing instance of the Firefox browser. See Recording a Web UI test by using a running browser instance on page 565 for details.

To record a test on Mozilla® Firefox® browser 54 or later, you need to select a user profile that is associated to the extension plug-in. For details, see Recording a Web UI test on page 545.

Recording a Web UI test

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

The type of application defines the recorder that can be used. You can use the following client application types that are supported for recording a Web UI test:

- Apple Safari on page 562
- Google Chrome on page 545
- Microsoft Edge on page 549
- Microsoft Internet Explorer on page 558
- Mozilla Firefox on page 553

Recording a Web UI test in Google Chrome

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

Before you begin

You must have completed the following tasks:

- Read and completed the tasks mentioned in Prerequisites for creating tests on page 534 if they apply to the test that you want to create.
- Created a project in the UI Test perspective. See Creating a Test Workbench project on page 38.

1. In the UI Test perspective, click **New > Test From Recording**.

   Alternatively, on the toolbar, click the **New Test From Recording** icon  .

   The **New Test From Recording** wizard is displayed.
2. Click **Create a test from a new recording** in the **Recording Session** screen.
3. Select **Web UI Test**.
4. Click **Recording Encryption Level**, and then select the encryption level, if you want to encrypt sensitive data in the application.
5. Click **Next**.

   The **Select Location** page is displayed.
6. Click to select a folder from the parent folder list.
7. Type a name for the test.
8. **Optional:** Select **Customize automatic data correlation** to correlate the test data by using automatic data correlation.
   This option is useful only if you are generating a HTTP test from the Web UI recsession file and want to apply data correlation to the HTTP test. Data correlation is not supported for a Web UI test.
9. Click **Next**.

   The **Select Client Application** page is displayed.
10. Select **Google Chrome**, and then click **Next**.

    The **Recorder Settings** page is displayed.
11. Set the following recording preferences:

Section	Option	Description
Security	The server requires a specific client certificate	Select this option if the server requires client SSL authentication, provide the client certificate for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Further, select one of the following options:

Section	Option	Description
		◦ Specify a certificate keystore file name and password to provide single certificate keystore. ◦ Click **Multiple certificates** if multiple certificates are required and specify a certificate keystore file name and password for each host name and port. ◦ Click **Single Smart Card** if you use smart card authentication for the application under test, and provide the alias and PIN code of the smart card. The smart card alias and PIN code is part of the smart card itself and is fetched from the reader. Click **Select** to choose an alias.
		Select the **Register the recorder root certificate authority** checkbox to install the recorder certificate to record a secured site. Before the recording starts, you are prompted to install the certificate. After the recording is stopped, you are prompted to uninstall the certificate. Select the **Keep the recorder root certificate authority after recording** checkbox to avoid multiple prompts for each recording.
Advanced	**Web UI recording > Private Mode(Default User profile)**	Select this option to launch the browser in private mode for recording the test. **Note:** Refer to prerequisites and enabling browser topics under Prerequisites for creating tests on page 534.
	Web UI recording > Launch browser with arguments	Select this option to specify arguments to be used when launching the browser. You can specify multiple browser launch arguments separated by a semicolon. **Note:** If you select this checkbox, the option remains selected for the subsequent test

Section	Option	Description
		recordings. You can then clear the checkbox if required.
	HTTP Traffic > Also record HTTP traffic	Use this option to include the performance aspect of the application in the Web UI recording session. This option is available only if both the HTTP and the Web UI extensions are installed. By using the **Also record HTTP traffic** option, you can include the HTTP traffic in the Web UI recording and later generate HTTP tests (in addition to the Web UI tests) from the recording. When you select **Also record HTTP traffic**, you can exclude traffic from specific destinations by clicking configure recording exceptions. You can also click change default value to specify preferred object identifiers.
	If the browser driver is incompatible, then Rational® Functional TesterHCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. And if the download fails, you must manually download the driver and place it in the configuration folder.	
Proxy Recorder	Proxy Type ◦ Auto ◦ HTTP ◦ SOCKS	Specify whether the proxy recorder uses **HTTP** or **SOCKS**. In most cases, select **Auto**. Select HTTP whenever authentication or proxies are required or when your application does not support SOCKS.
Advanced Security	The client requires a specific server certificate • SSL Encryption: Override browser settings • Accept SSL 3.0 • Accept TLS 1.0 • Accept TLS 1.1 • Accept TLS 1.2	Select this option to specify whether to use an HTTP or SOCKS proxy recorder to review and edit network connection settings that the browser uses or to specify advanced SSL authentication settings. Specify advanced SSL authentication settings. If you are using the SOCKS recorder, the `RptCertificate.jks` certificate is used by default. Select The client requires a specific server certificate and click Add to specify the server hostname, port, certificate

Section	Option	Description
		database path, and the certificate database password for each website that you are planning to test. If you select Generate certificate, for any IP address received by the SOCK proxy that is resolved by one the server hostnames that you listed, the SOCKS proxy uses a generated certificate signed by the RPT certificate authority, thereby ensuring a smooth recording. If you select **Override browser settings**, select **Accept SSL 3.0**, or **Accept TLS 1.0**, or both.

12. Click **Finish**.
    The Welcome page is displayed with the following details:
    - Version of the web browser.
    - Web applications that you added to test.
13. In the browser address field, type the address of the web application to test.

    Note: If you enter the address of a secure website (one that starts with `https:`), your browser might display a security alert. Depending on the security certificate for the site, you might be required to accept a security risk to proceed with the recording.

14. After you finish the user tasks in the browser, stop the recorder. You can stop the recorder by closing the web application under test or by clicking the **Stop** icon in the **Recording Control** view.
15. In the **Data Correlation and Transformation** page, set the data correlation options as appropriate and click **Finish**.
    This page appears if you have selected the **Customize automatic data correlation** option.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording a Web UI test in Microsoft Edge

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

Before you begin

You must have completed the following tasks:

- Read and completed the tasks mentioned in Prerequisites for creating tests on page 534 if they apply to the test that you want to create.
- Created a project in the UI Test perspective. See Creating a Test Workbench project on page 38.

1. In the UI Test perspective, click **New > Test From Recording**.

   Alternatively, on the toolbar, click the **New Test From Recording** icon  .
2. Click **Create a test from a new recording**.
3. Select **Web UI Test**.
4. Click **Recording Encryption Level**, and then select the encryption level, if you want to encrypt sensitive data in the application.
5. Click **Next**.

   The **Select Location** page is displayed.
6. Click to select a folder from the parent folder list.
7. Type a name for the test.
8. **Optional:** Select **Customize automatic data correlation** to correlate the test data by using automatic data correlation.
   This option is useful only if you are generating a HTTP test from the Web UI recsession file and want to apply data correlation to the HTTP test. Data correlation is not supported for a Web UI test.
9. Click **Next**.

   The **Select Client Application** page is displayed.
10. Select **Microsoft Edge**, and then click **Next**.

    The **Recorder Settings** dialog is displayed.
11. Set the following recording preferences:

Section	Option	Description
Security	The server requires a specific client certificate	Select this option if the server requires client SSL authentication, provide the client certificate for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Further, select one of the following options: ◦ Specify a certificate keystore file name and password to provide single certificate keystore. ◦ Click **Multiple certificates** if multiple certificates are required and specify a certificate keystore file name and password for each host name and port.

Section	Option	Description
		◦ Click **Single Smart Card** if you use smart card authentication for the application under test, and provide the alias and PIN code of the smart card. The smart card alias and PIN code is part of the smart card itself and is fetched from the reader. Click **Select** to choose an alias.
		Select the **Register the recorder root certificate authority** checkbox to install the recorder certificate to record a secured site. Before the recording starts, you are prompted to install the certificate. After the recording is stopped, you are prompted to uninstall the certificate. Select the **Keep the recorder root certificate authority after recording** checkbox to avoid multiple prompts for each recording.
Advanced	**Web UI recording > Private Mode(Default User profile)**	Select this option to launch the browser in private mode for recording the test. **Note:** Refer to prerequisites and enabling browser topics under Prerequisites for creating tests on page 534.
	Web UI recording > Launch browser with arguments	Select this option to specify arguments to be used when launching the browser. You can specify multiple browser launch arguments separated by a semicolon. **Note:** If you select this checkbox, the option remains selected for the subsequent test recordings. You can then clear the checkbox if required.
	HTTP Traffic > Also record HTTP traffic	Use this option to include the performance aspect of the application in the Web UI recording session. This option is available only if both the HTTP and the Web UI extensions are installed.

Section	Option	Description
		By using the **Also record HTTP traffic** option, you can include the HTTP traffic in the Web UI recording and later generate HTTP tests (in addition to the Web UI tests) from the recording. When you select **Also record HTTP traffic**, you can exclude traffic from specific destinations by clicking configure recording exceptions. You can also click change default value to specify preferred object identifiers.
	If the browser driver is incompatible, then Rational® Functional TesterHCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. And if the download fails, you must manually download the driver and place it in the configuration folder.	
Proxy Recorder	Proxy Type ◦ Auto ◦ HTTP ◦ SOCKS	Specify whether the proxy recorder uses **HTTP** or **SOCKS**. In most cases, select **Auto**. Select HTTP whenever authentication or proxies are required or when your application does not support SOCKS.
Connection to the Network	Override browser settings • Direct connection to the network • HTTP proxy • SOCKS proxy • Automatic configuration script • System settings	Review the proxy settings that are retrieved from the Mozilla Firefox configuration. If necessary, select **Override browser settings** to specify an HTTP or SOCKS proxy, to point to a Proxy Auto-Configuration (PAC) file, or to use system settings.
Advanced Security	The client requires a specific server certificate • SSL Encryption: Override browser settings • Accept SSL 3.0 • Accept TLS 1.0 • Accept TLS 1.1	Select this option to specify whether to use an HTTP or SOCKS proxy recorder to review and edit network connection settings that the browser uses or to specify advanced SSL authentication settings. Specify advanced SSL authentication settings. If you are using the SOCKS recorder, the `RptCertificate.jks` certificate is used by default. Select The

Section	Option	Description
	• Accept TLS 1.2	client requires a specific server certificate and click Add to specify the server hostname, port, certificate database path, and the certificate database password for each website that you are planning to test. If you select Generate certificate, for any IP address received by the SOCK proxy that is resolved by one the server hostnames that you listed, the SOCKS proxy uses a generated certificate signed by the RPT certificate authority, thereby ensuring a smooth recording. If you select **Override browser settings**, select **Accept SSL 3.0**, or **Accept TLS 1.0**, or both.

12. Click **Finish**.
    The Welcome page is displayed with the following details:
    - Version of the web browser.
    - Web applications that you added to test.
13. In the browser address field, type the address of the web application to test.

    Note: If you enter the address of a secure website (one that starts with `https:`), your browser might display a security alert. Depending on the security certificate for the site, you might be required to accept a security risk to proceed with the recording.

14. After you finish the user tasks in the browser, stop the recorder. You can stop the recorder by closing the web application under test or by clicking the **Stop** icon in the **Recording Control** view.
15. In the **Data Correlation and Transformation** page, set the data correlation options as appropriate and click **Finish**.
    This page appears if you have selected the **Customize automatic data correlation** option.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording a Web UI test in Mozilla Firefox

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

Before you begin

You must have completed the following tasks:

- Read and completed the tasks mentioned in Prerequisites for creating tests on page 534 if they apply to the test that you want to create.
- Created a project in the UI Test perspective. See Creating a Test Workbench project on page 38.

1. In the UI Test perspective, click **New > Test From Recording**.

   Alternatively, on the toolbar, click the **New Test From Recording** icon  .
2. Click **Create a test from a new recording**.
3. Select **Web UI Test**.
4. Click **Recording Encryption Level**, and then select the encryption level, if you want to encrypt sensitive data in the application.
5. Click **Next**.

   The **Select Location** page is displayed.
6. Click to select a folder from the parent folder list.
7. Type a name for the test.
8. **Optional:** Select **Customize automatic data correlation** to correlate the test data by using automatic data correlation.
   This option is useful only if you are generating a HTTP test from the Web UI recsession file and want to apply data correlation to the HTTP test. Data correlation is not supported for a Web UI test.
9. Click **Next**.

   The **Select Client Application** page is displayed.
10. Select **Mozilla Firefox**, and then click **Next**.

    The **Recorder Settings** dialog is displayed.
11. Set the following recording preferences:

Section	Option	Description
Security	The server requires a specific client certificate	Select this option if the server requires client SSL authentication, provide the client certificate for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Further, select one of the following options: ◦ Specify a certificate keystore file name and password to provide single certificate keystore. ◦ Click **Multiple certificates** if multiple certificates are required and specify a certificate

Section	Option	Description
		keystore file name and password for each host name and port. ◦ Click **Single Smart Card** if you use smart card authentication for the application under test, and provide the alias and PIN code of the smart card. The smart card alias and PIN code is part of the smart card itself and is fetched from the reader. Click **Select** to choose an alias. To ease HTTPS recording, save the recorder root certificate authority and add it to the trusted authorities.
Firefox profile	Use an alternate Firefox profile	The recorder uses the default Firefox profile. This can cause problems during recording if your Firefox settings generate a behavior that you do not want to record (for example: extensions, toolbars, home pages, or recorded passwords and cookies). This option starts Firefox® without any bookmarks, plug-ins, or toolbars that might be associated with your usual profile. Further, you can specify one of the following profile options: ◦ Select default to start the web browser with default user profile. ◦ Select default-release to start the web browser with a new profile. ◦ Select Use a temporary Firefox profile to start the web browser with a new profile. **Note:** Do not select this option if you want to record the test in private mode.
Advanced	**Web UI recording > Private Mode(Default User profile)**	Select this option to launch the browser in private mode for recording the test.

Section	Option	Description
		Note: Refer to prerequisites and enabling browser topics under Prerequisites for creating tests on page 534.
	Web UI recording > Launch browser with arguments	Select this option to specify arguments to be used when launching the browser. You can specify multiple browser launch arguments separated by a semicolon. **Note:** If you select this checkbox, the option remains selected for the subsequent test recordings. You can then clear the checkbox if required.
	HTTP Traffic > Also record HTTP traffic	Use this option to include the performance aspect of the application in the Web UI recording session. This option is available only if both the HTTP and the Web UI extensions are installed. By using the **Also record HTTP traffic** option, you can include the HTTP traffic in the Web UI recording and later generate HTTP tests (in addition to the Web UI tests) from the recording. When you select **Also record HTTP traffic**, you can exclude traffic from specific destinations by clicking configure recording exceptions. You can also click change default value to specify preferred object identifiers.
	If the browser driver is incompatible, then Rational® Functional TesterHCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. And if the download fails, you must manually download the driver and place it in the configuration folder.	
Proxy Recorder	Proxy Type	Specify whether the proxy recorder uses **HTTP** or **SOCKS**. In most cases, select **Auto**. Select HTTP

Section	Option	Description
	◦ Auto ◦ HTTP ◦ SOCKS	whenever authentication or proxies are required or when your application does not support SOCKS.
Connection to the Network	Override browser settings • Direct connection to the network • HTTP proxy • SOCKS proxy • Automatic configuration script • System settings	Review the proxy settings that are retrieved from the Mozilla Firefox configuration. If necessary, select **Override browser settings** to specify an HTTP or SOCKS proxy, to point to a Proxy Auto-Configuration (PAC) file, or to use system settings.
Advanced Security	The client requires a specific server certificate • SSL Encryption: Override browser settings • Accept SSL 3.0 • Accept TLS 1.0 • Accept TLS 1.1 • Accept TLS 1.2	Select this option to specify whether to use an HTTP or SOCKS proxy recorder to review and edit network connection settings that the browser uses or to specify advanced SSL authentication settings. Specify advanced SSL authentication settings. If you are using the SOCKS recorder, the `RptCertificate.jks` certificate is used by default. Select The client requires a specific server certificate and click Add to specify the server hostname, port, certificate database path, and the certificate database password for each website that you are planning to test. If you select Generate certificate, for any IP address received by the SOCK proxy that is resolved by one the server hostnames that you listed, the SOCKS proxy uses a generated certificate signed by the RPT certificate authority, thereby ensuring a smooth recording. If you select **Override browser settings**, select **Accept SSL 3.0**, or **Accept TLS 1.0**, or both.

12. Click **Finish**.
    The Welcome page is displayed with the following details:
    ◦ Version of the web browser.
    ◦ Web applications that you added to test.
13. In the browser address field, type the address of the web application to test.

Note:

If you enter the address of a secure website (one that starts with `https:`), your browser might display a security alert. Depending on the security certificate for the site, you might be required to accept a security risk to proceed with the recording.

You can change the browser zoom level during recording in Mozilla Firefox browsers. In some operating systems, zooming during recording may not work as expected. As a workaround to this problem, in the browser, click **View > Zoom > Zoom Text Only**.

14. After you finish the user tasks in the browser, stop the recorder. You can stop the recorder by closing the web application under test or by clicking the **Stop** icon in the **Recording Control** view.
15. In the **Data Correlation and Transformation** page, set the data correlation options as appropriate and click **Finish**.
    This page appears if you have selected the **Customize automatic data correlation** option.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording a Web UI test in Internet Explorer

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

Before you begin

You must have completed the following tasks:

- Read and completed the tasks mentioned in Prerequisites for creating tests on page 534 if they apply to the test that you want to create.
- Created a project in the UI Test perspective. See Creating a Test Workbench project on page 38.

1. In the UI Test perspective, click **New > Test From Recording**.

   Alternatively, on the toolbar, click the **New Test From Recording** icon .
2. Click **Create a test from a new recording**.
3. Select **Web UI Test**.
4. Click **Recording Encryption Level**, and then select the encryption level, if you want to encrypt sensitive data in the application.
5. Click **Next**.

The **Select Location** page is displayed.

6. Click to select a folder from the parent folder list.
7. Type a name for the test.
8. **Optional:** Select **Customize automatic data correlation** to correlate the test data by using automatic data correlation.
   This option is useful only if you are generating a HTTP test from the Web UI recsession file and want to apply data correlation to the HTTP test. Data correlation is not supported for a Web UI test.
9. Click **Next**.

   The **Select Client Application** page is displayed.

10. Select **Microsoft Internet Explorer**, and then click **Next**.

    The **Recorder Settings** dialog is displayed.

11. Set the following recording preferences:

Section	Option	Description
Security	The server requires a specific client certificate	Select this option if the server requires client SSL authentication, provide the client certificate for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Further, select one of the following options: ◦ Specify a certificate keystore file name and password to provide single certificate keystore. ◦ Click **Multiple certificates** if multiple certificates are required and specify a certificate keystore file name and password for each host name and port. ◦ Click **Single Smart Card** if you use smart card authentication for the application under test, and provide the alias and PIN code of the smart card. The smart card alias and PIN code is part of the smart card itself and is fetched from the reader. Click **Select** to choose an alias.
		Select the **Register the recorder root certificate authority** checkbox to install the recorder certificate to record a secured site. Before the recording starts, you are prompted to install the certificate. After the

Section	Option	Description
		recording is stopped, you are prompted to uninstall the certificate. Select the **Keep the recorder root certificate authority after recording** checkbox to avoid multiple prompts for each recording.
Advanced	**HTTP Traffic > Also record HTTP traffic**	Use this option to include the performance aspect of the application in the Web UI recording session. This option is available only if both the HTTP and the Web UI extensions are installed. By using the **Also record HTTP traffic** option, you can include the HTTP traffic in the Web UI recording and later generate HTTP tests (in addition to the Web UI tests) from the recording. When you select **Also record HTTP traffic**, you can exclude traffic from specific destinations by clicking configure recording exceptions. You can also click change default value to specify preferred object identifiers.
	If the browser driver is incompatible, then Rational® Functional TesterHCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. And if the download fails, you must manually download the driver and place it in the configuration folder.	
Proxy Recorder	Proxy Type ◦ Auto ◦ HTTP ◦ SOCKS	Specify whether the proxy recorder uses **HTTP** or **SOCKS**. In most cases, select **Auto**. Select HTTP whenever authentication or proxies are required or when your application does not support SOCKS.
Connection to the Network	Override browser settings • Direct connection to the network • HTTP proxy • SOCKS proxy • Automatic configuration script	Review the proxy settings that are retrieved from the Mozilla Firefox configuration. If necessary, select **Override browser settings** to specify an HTTP or SOCKS proxy, to point to a Proxy Auto-Configuration (PAC) file, or to use system settings.

Section	Option	Description
	• System settings	
Advanced Security	The client requires a specific server certificate • SSL Encryption: Override browser settings • Accept SSL 3.0 • Accept TLS 1.0 • Accept TLS 1.1 • Accept TLS 1.2	Select this option to specify whether to use an HTTP or SOCKS proxy recorder to review and edit network connection settings that the browser uses or to specify advanced SSL authentication settings. Specify advanced SSL authentication settings. If you are using the SOCKS recorder, the `RptCertificate.jks` certificate is used by default. Select The client requires a specific server certificate and click Add to specify the server hostname, port, certificate database path, and the certificate database password for each website that you are planning to test. If you select Generate certificate, for any IP address received by the SOCK proxy that is resolved by one the server hostnames that you listed, the SOCKS proxy uses a generated certificate signed by the RPT certificate authority, thereby ensuring a smooth recording. If you select **Override browser settings**, select **Accept SSL 3.0**, or **Accept TLS 1.0**, or both.

12. Click **Finish**.
    The Welcome page is displayed with the following details:
    - Version of the web browser.
    - Web applications that you added to test.
13. In the browser address field, type the address of the web application to test.

    Note: If you enter the address of a secure website (one that starts with `https:`), your browser might display a security alert. Depending on the security certificate for the site, you might be required to accept a security risk to proceed with the recording.

14. After you finish the user tasks in the browser, stop the recorder. You can stop the recorder by closing the web application under test or by clicking the **Stop** icon in the **Recording Control** view.
15. In the **Data Correlation and Transformation** page, set the data correlation options as appropriate and click **Finish**.
    This page appears if you have selected the **Customize automatic data correlation** option.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording a Web UI test in Apple Safari

You create a Web UI test in the UI Test perspective. Rational® Functional TesterHCL OneTest™ UI automatically enables web browsers and configures the test environment before you start recording tests for web applications.

Before you begin

You must have completed the following tasks:

- Read and completed the tasks mentioned in Prerequisites for creating tests on page 534 if they apply to the test that you want to create.
- Created a project in the UI Test perspective. See Creating a Test Workbench project on page 38.

1. In the UI Test perspective, click **New > Test From Recording**.

   Alternatively, on the toolbar, click the **New Test From Recording** icon  .
2. Click **Create a test from a new recording**.
3. Select **Web UI Test**.
4. Click **Recording Encryption Level**, and then select the encryption level, if you want to encrypt sensitive data in the application.
5. Click **Next**.

   The **Select Location** page is displayed.
6. Click to select a folder from the parent folder list.
7. Type a name for the test.
8. **Optional:** Select **Customize automatic data correlation** to correlate the test data by using automatic data correlation.
   This option is useful only if you are generating a HTTP test from the Web UI recsession file and want to apply data correlation to the HTTP test. Data correlation is not supported for a Web UI test.
9. Click **Next**.

   The **Select Client Application** page is displayed.
10. Select **Apple Safari**, and then click **Next**.

    The **Recorder Settings** dialog is displayed.
11. Set the following recording preferences:

Section	Option	Description
Security	The server requires a specific client certificate	Select this option if the server requires client SSL authentication, provide the client certificate for the proxy recorder to be authenticated by the server as though the proxy recorder were the client. Further, select one of the following options: ◦ Specify a certificate keystore file name and password to provide single certificate keystore. ◦ Click **Multiple certificates** if multiple certificates are required and specify a certificate keystore file name and password for each host name and port. ◦ Click **Single Smart Card** if you use smart card authentication for the application under test, and provide the alias and PIN code of the smart card. The smart card alias and PIN code is part of the smart card itself and is fetched from the reader. Click **Select** to choose an alias.
		Select the **Register the recorder root certificate authority** checkbox to install the recorder certificate to record a secured site. Before the recording starts, you are prompted to install the certificate. After the recording is stopped, you are prompted to uninstall the certificate. Select the **Keep the recorder root certificate authority after recording** checkbox to avoid multiple prompts for each recording.
Advanced	**HTTP Traffic > Also record HTTP traffic**	Use this option to include the performance aspect of the application in the Web UI recording session. This option is available only if both the HTTP and the Web UI extensions are installed. By using the **Also record HTTP traffic** option, you can include the HTTP traffic in the Web UI recording and later generate HTTP tests (in addition to the Web UI tests) from the recording.

Section	Option	Description
		When you select **Also record HTTP traffic**, you can exclude traffic from specific destinations by clicking configure recording exceptions. You can also click change default value to specify preferred object identifiers.
	If the browser driver is incompatible, then Rational® Functional TesterHCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. And if the download fails, you must manually download the driver and place it in the configuration folder.	
Proxy Recorder	Proxy Type ◦ Auto ◦ HTTP ◦ SOCKS	Specify whether the proxy recorder uses **HTTP** or **SOCKS**. In most cases, select **Auto**. Select HTTP whenever authentication or proxies are required or when your application does not support SOCKS.
Connection to the Network	Override browser settings • Direct connection to the network • HTTP proxy • SOCKS proxy • Automatic configuration script • System settings	Review the proxy settings that are retrieved from the Mozilla Firefox configuration. If necessary, select **Override browser settings** to specify an HTTP or SOCKS proxy, to point to a Proxy Auto-Configuration (PAC) file, or to use system settings.
Advanced Security	The client requires a specific server certificate • SSL Encryption: Override browser settings • Accept SSL 3.0 • Accept TLS 1.0 • Accept TLS 1.1 • Accept TLS 1.2	Select this option to specify whether to use an HTTP or SOCKS proxy recorder to review and edit network connection settings that the browser uses or to specify advanced SSL authentication settings. Specify advanced SSL authentication settings. If you are using the SOCKS recorder, the `RptCertificate.jks` certificate is used by default. Select The client requires a specific server certificate and click Add to specify the server hostname, port, certificate database path, and the certificate database password for each website that you are planning to test. If you select Generate certificate, for any IP address re-

Section	Option	Description
		ceived by the SOCK proxy that is resolved by one the server hostnames that you listed, the SOCKS proxy uses a generated certificate signed by the RPT certificate authority, thereby ensuring a smooth recording. If you select **Override browser settings**, select **Accept SSL 3.0**, or **Accept TLS 1.0**, or both.

12. Click **Finish**.
    The Welcome page is displayed with the following details:
    - Version of the web browser.
    - Web applications that you added to test.
13. In the browser address field, type the address of the web application to test.

    Note: If you enter the address of a secure website (one that starts with `https:`), your browser might display a security alert. Depending on the security certificate for the site, you might be required to accept a security risk to proceed with the recording.

14. After you finish the user tasks in the browser, stop the recorder. You can stop the recorder by closing the web application under test or by clicking the **Stop** icon in the **Recording Control** view.
15. In the **Data Correlation and Transformation** page, set the data correlation options as appropriate and click **Finish**.
    This page appears if you have selected the **Customize automatic data correlation** option.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording a Web UI test by using a running browser instance

You can create a Web UI test of a web application from the UI Test perspective using a browser that is already running locally. For example, if you have already recorded some part of a web application, you can go back at a later time to resume the recording Support is provided for Chrome and Firefox on Windows and Linux computers and for Safari, Chrome and Firefox on Macintosh computers. Internet Explorer is not supported.

Before you begin

- Install the Web UI extensions for the browsers that you plan to use for testing. For instructions, see Enabling Google Chrome for Web UI testing on page 540, Enabling Mozilla Firefox for Web UI testing on page 543, Enabling the Apple Safari browser to perform Web UI tests on macOS on page 538, and Enabling Microsoft Edge for Web UI testing on page 542.
- When recording a test, wait for each page to load completely. This waiting time does not affect performance results, because you can remove extra waiting time (think time) when you play back the test.
- Do not change any browser preferences, including JavaScript settings. Recording and playing back Web UI scripts in a browser requires that JavaScript be enabled.
- To record a test that emulates a web application on a mobile device, see Recording a test with Google Chrome Device Mode on page 568.

About this task

Video: Learn this functionality with the help of this video.

1. Start a supported browser session and type the URL of the web application to test.
2. In the UI Test perspective, click **New > Test From Recording**. Alternatively, on the toolbar, click the **New Test From Recording** icon .
3. Click **Create a test from a new recording**. Select **Web UI Test**. If you are recording sensitive data, click **Recording Encryption Level** and select the encryption level to record. Click **Next**.
4. **Optional:** If you did not create a test project earlier, click the **Create the parent folder** icon to create a test project. For more information, see Creating project on page 38.
5. Type a name for the test and click **Next**.
6. In the Select Client Application page, select **Running browser instance** and click **Next**.
7. Select the web application to test from the list of running browser instances and click **Finish**.
8. Return to the browser and record the test.
9. After you finish the user tasks in the browser, stop the recorder by clicking the **Stop** icon in the **Recording Control** view.
   Result

   After you stop the recorder, the test is generated from the recording.
10. Optional: Open the test for editing.

What to do next

When the test is generated, you can edit it in the test editor. For information, see Editing Web UI tests on page 573.

To create variable data for the test, you can use the dataset candidates suggested by the workbench when you first open the generated test. For more information, see Viewing dataset candidates when you open a test on page 652.

Recording steps for an existing Web UI test

If you already have a test, you can add more steps to it at a desired location by initiating the recording from a step or from the application node. For instance, if the task flow has changed in the new version of the application that you

recorded and you want to update the existing test script, start the recording from the step after which the new steps must be added. You can also initiate the recording from the Launch application or In application nodes of the test.

1. Open the test script in which to add steps from a new recording.
2. Right-click the step after which the new steps must be added and click **Insert > Steps from Recording (web)**. You can also start the recording from the "Launch application", "In application", and "Window" nodes of a test script. To do that, right-click the node and click **Add > Steps from Recording (web)**.
3. In the **Select Client Application** page of the recording wizard, select a web browser or click **Running Browser Instance**. Click **Next**.

   When you select a browser, Rational® Functional TesterHCL OneTest™ UI configures and starts the browser for you to start the recording from the first page of the application. When you select **Running Browser Instance**, you start recording the application that is already running in a browser from the current state of the application so that you do not need to start recording from the first page of the application.
4. Complete one of the following steps:
   a. If you selected a web browser, follow the next set of instructions in Recording a Web UI test on page 545.
   b. If you selected **Running Browser Instance**, follow the next set of instructions in Recording steps for an existing Web UI test on page 566.
5. After the recording is done, the test is generated. Click **Close**. The newly recorded steps are added after the step from where you initiated the recording. If you initiated the recording from the application node, the new steps are added after the application node or after the last step in the application node. All the new steps are highlighted. To save the changes, click **Save**.

What to do next

You can enhance the test. See Editing Web UI tests on page 573.

Recording a Web UI test immediately after the previous playback

When you want to continue the testing after the previous playback, you can choose to record a new Web UI test by using the **Allow recording after playback** option.

Before you begin

You must have created one or more Web UI tests. See Creating Web UI tests on page 532.

About this task

You can keep the browser active after the playback is complete for a Web UI test, and then start to record from the point where the playback stopped.

1. Start Rational® Functional TesterHCL OneTest™ UI and perform the following steps:
   a. Click **Window > Preferences > Test > Test Execution > UI Test Playback > Desktop** tab.
   b. Select the **Allow recording after playback** option under **Playback options** to continue recording in the same browser after the playback.
   c. Click **Apply and Close**.

2. Open the Web UI test for playback from the **Test Navigator** pane.

   The test is displayed.

3. **Optional:** Remove or disable the step if the close browser action is listed in the script.
   The close browser action is recorded when you intentionally close the browser during the recording process. However, you can choose to stop a recording and not record the close browser action.
4. Click the **Run Test** icon.
5. Click the **UI Test** perspective after the test run is complete.
6. Click **New > Test From Recording** or click the **New Test From Recording** icon on the toolbar in the UI Test perspective.
7. Click **Create a test from a new recording**, and then select **Web UI Test**.
8. Click **Next**.
9. Type a name for the test, and then click **Next**.
10. Select **Running browser instance**, and then click **Next** in the **Select Client Application** page.
11. Select the web application that you tested from the list of running browser instances and click **Finish**.
12. Record the test from the point where the previous playback had stopped.
13. Stop the recorder by clicking the **Stop** icon in the **Recording Control** view after you finish the user tasks in the browser.
    The test is generated from the recording.

Results

You have recorded a Web UI test immediately after the previous playback is complete.

Related information

Recording a Web UI test by using a running browser instance on page 565

Recording steps for an existing Web UI test on page 566

Recording a test with Google Chrome Device Mode

You can use the Google Chrome Device Mode feature to emulate tests of web applications on mobile devices.

Before you begin

Read and been familiar with the following tasks:

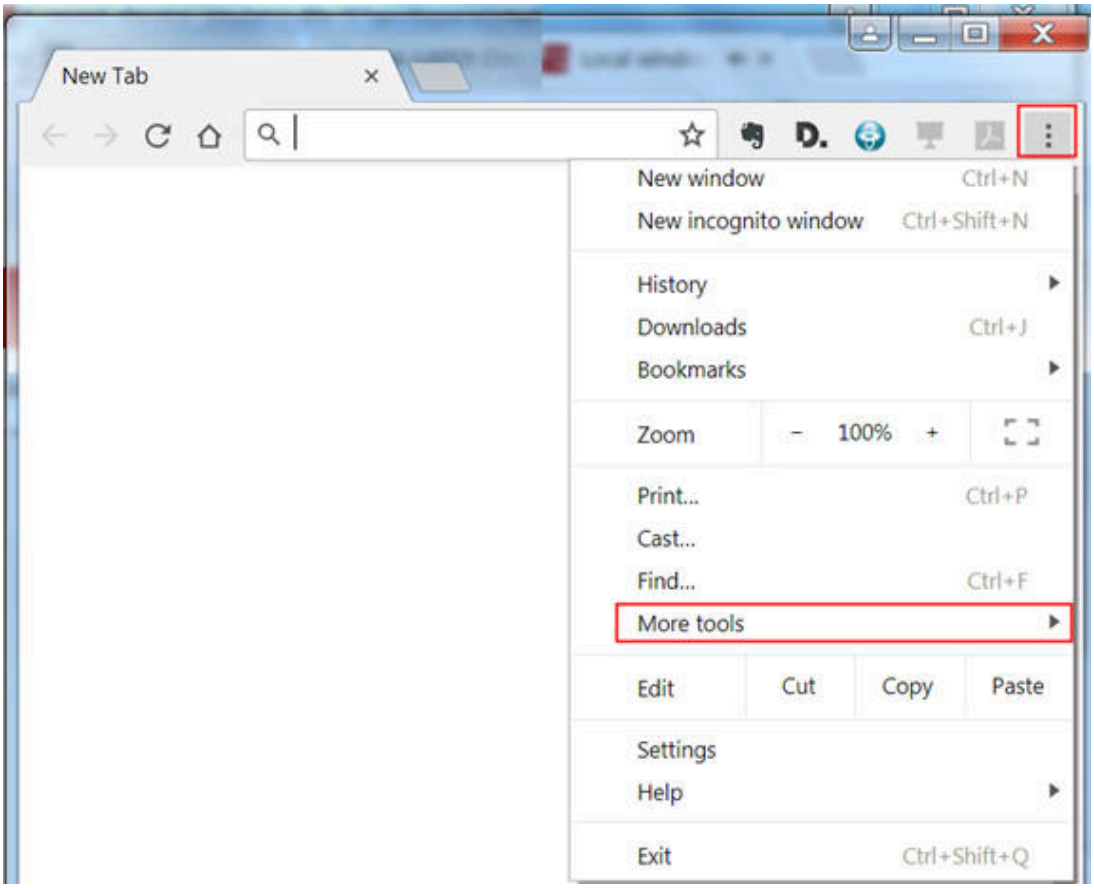
- Recording a Web UI test on page 545
- Recording a Web UI test by using a running browser instance on page 565

About this task

You can enable the Device Mode feature in Google Chrome to use it as a browser on a mobile device. Then, you can record a Web UI test using one of the devices listed in Google Chrome. The test is recorded as though it was recorded on the device and later, when you run the test, the test runs as though it is run on the device.

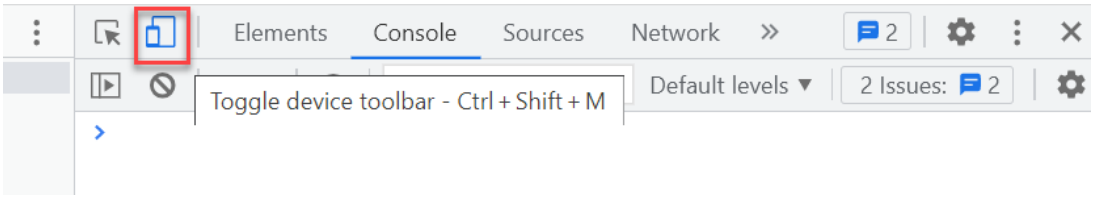
1. Start a new Web UI recording with the Google Chrome browser. Alternatively, you can record the test by using a running instance of Chrome.
2. In Chrome, configure the Chrome Device Mode feature to emulate a particular mobile device.

    a. Select **More Tools > Developer Tools** by clicking the menu in the upper-right corner of the Google Chrome window.



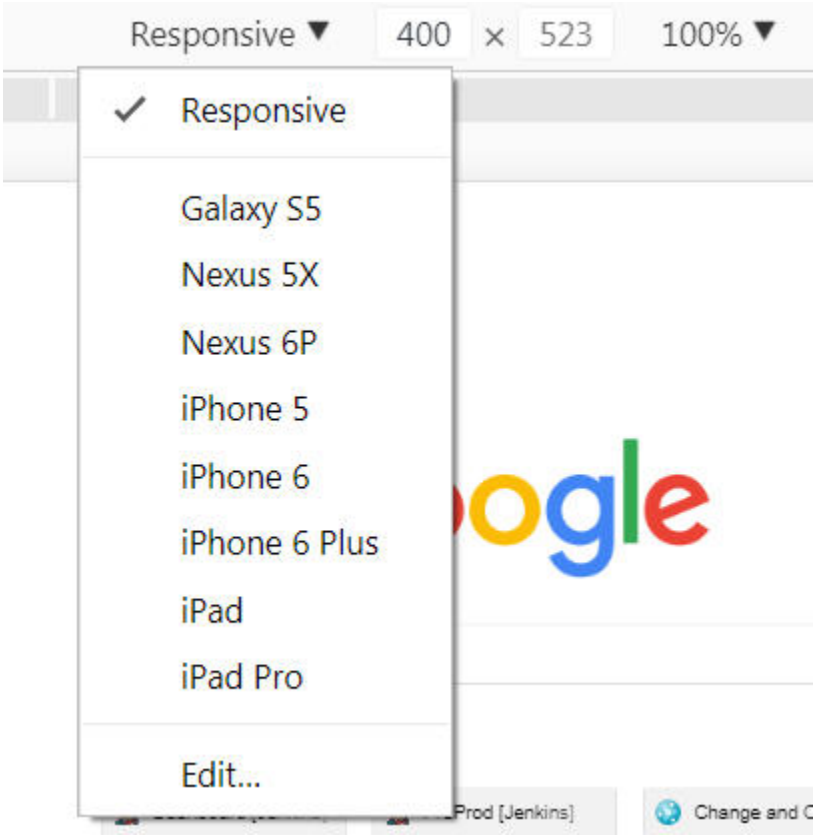
    You can also open Developer Tools by pressing the `Cntl+Shift+I` keys or the `F12` key (Windows only).

    b. Click the **Toggle device toolbar** button.



    c. Select the device that you want to emulate from the **Responsive** menu.

For example, select **iPhone 6**. Also, ensure that the zoom level is set to 100%.



3. Load the web application under test to allow it to adjust and perform according to the device mode that is selected.
4. Continue to record the test.

Annotating a test during recording

You can add comments, add transactions, or change a page name while you record a test. The advantage of adding these elements during (rather than after) recording is that you can place the annotations in the test exactly where you want. In addition, because annotations are part of the recorded test, they are regenerated when you regenerate the test. You can also insert split points into a test during record.

1. Start recording the test. The **Recorder Test Annotations** toolbar opens near the top of the screen.
2. Click the appropriate icon.

   You can use the **Recorder Test Annotations** toolbar to add comments, record synchronizations, or take screen captures during the recording.

- To add a comment to the recorded test, click the **Insert comment** icon  . You are prompted for a comment.
- To add a screen capture to the recorded test, click the **Capture screen** icon  . Screen and window captures make your tests easier to read and help you visualize the recorded test. You can change the settings for screen captures and add a comment to the image.
- To manually add a transaction folder to the recording, click the **Start Transaction** icon  and **Stop Transaction** icon  to start and stop the transaction. Transactions can be nested.
- To insert a split point into the recorded test, click the **Split point** icon  . With split points, you can generate multiple tests from a single recording, which you can replay in a different order with a schedule. See Splitting a test during recording on page 571 for more information about splitting a test.
- When recording an HTTP test, to change the page name, click the **Change page name** icon  . In the resulting test, the page element in the test editor uses the new name, however the original name is preserved in the **Page Title Verification Point** area so that page title verification points still work correctly.

3. Close the client program to stop the recording.
4. If you inserted a split point during the recording, on the **Destination** page, in the **Test Generation** wizard, specify the location for the split test or merge the split recordings together.

Results

The test is generated with the comments, transactions, and page names that you added.

Splitting a test during recording

You can insert split points when you record a test. Split points allow you to generate multiple tests from a single recording that you can replay in a different order with a schedule.

To split a test during recording:

1. Start recording the test. The **Recorder Test Annotations** toolbar opens near the top of the screen.
2. To insert a split point into the recorded test, click the  **Split point** button. . The **Insert Split Point** window is displayed.
3. Type a name for this section of the test and click **OK**. You are naming the previous section of the test, not the upcoming section of the test.
   Repeat this step between recorded user actions as needed to split tests.
4. After you finish performing the user tasks in the client program, stop the recorder. You can do this by closing the client program or by clicking the **Stop**  button in the **Recorder Control** view.
   If you changed the network settings of the client program as described in step 8, you can revert them to the default settings before closing the program.
   Result
   The **Generate Service Test** wizard opens.
5. On the **Destination** page, specify the location for the split test or merge the split recordings together:

- In **Location**, click **Browse** to specify the folder where the split tests are generated.
- Type a **Test prefix** that will be appended to the name of each split test. Leave blank if you do not want the split test names to have a prefix.
- In the split test list, mark the split tests that you want to generate. Click **Select All** to generate all split tests or **Unselect All** to clear the list.
- To merge several split tests into a single test, multi-select the tests that you want to merge by holding the **Shift** key and click the **Merge** button.

6. Click **Finish**.

Results

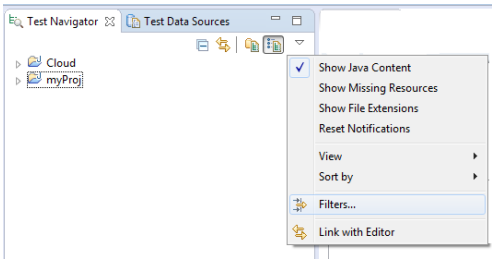
The tests are generated using the test names that you specified.

Putting test assets under source control

Use version-control software, such as Rational® Team Concert™, to put the test assets under source control.

If you use a version-control software, such as or Rational® Team Concert™, put the following assets under source control to share them with the members of your team that use the same source-control product. Put the different project assets in separate folders under the main project.

Asset	File name extension	Comments
Projects	`.project`	This enables the project to be seen and imported by another user.
Manifest and build files	MANIFEST.MF, build.properties	These are project files that are checked in.
Recording	`.recdata, .recsession`	The recording data and the session information will be part of these files.
Tests	`.testsuite`	The test script.
Managed Application	.ma	This is the application under test that is instrumented with the code.
Custom code	`.java`	Put any custom code that you have written for a test under the source control. Put this code under the `src` folder for the project; for example, in `src\custom`. This code must be versioned as a single logical unit with the test that includes it (that is, the code and the test should be versioned together).
JavaScript	`.js`	Put any JavaScript file that you have used for a test under the source control.

Asset	File name extension	Comments
Datasets	`.dataset`	Put any Dataset that you have used for a test under the source control.
Results	`.xmoebreport, .stats, .rstats, .pstats, and .mstats`	These are results assets and contain the data that is used to create reports and should be under source control. Some of these assets are not visible in the Test Navigator. To see these assets, in the Test Navigator view, click the drop-down icon and select **Filters** and clear the **Statistics Stores** check box.  You can perform the check out and check in operations from this view. Store the results in a separate results folder, which you can specify when running a schedule or test.

Follow these guidelines:

- During a session, keep schedules and tests checked out for easy editing. When you exit Eclipse, you are prompted to check them back in.
- Put test logs (files with the `.execution` file name extension) under source control, in the results folder.
- Do not put the class path file under source control.

Editing Web UI tests

With the test editor, you can view and customize a Web UI test.

The test editor displays the test scripts. The edited test displays the list of actions and UI elements recorded during the recording phase. Actions are represented in natural language, which allows you to modify the test manually.

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, **User Action Details**, displays details about the currently selected action in the test script. In this area you can select a graphic object, an action related to the object, and its location. You can also define a time out and users think time to execute your test. Think time is a statistical emulation of the amount of time that you spend on reading or thinking, before performing an action. It is the programmatically-calculated time delay that is observed for users.

When actions are selected in the **Test Contents** list, the UI Test Data is automatically synchronized to display the screen captures of the user interface of the app during the recording. You can use the UI Test Data to select user interface (UI) elements and add some verification points, or variables and modify steps in the test with simplified scripts. Or you can create or modify a set of steps manually directly in the test script.

You can add datasets, test variables, or verification points in your script.

Inserting browser navigation actions in a Web UI test

While editing a Web UI test, you can add or insert browser navigation actions, such as Back, Forward, Close, Maximize, Refresh, Restore, GoToUrl, and several others.

Before you begin

Ensure that you have created a Web UI test from a recording and have the test script open in the test editor.

1. Open a Web UI test script and in the **Test Contents** area, click in the launch app node where you want the navigation action to be inserted.
2. Click the **Insert** button and select **Navigation action (Web)**. Another way is to right-click the selection or click **Options** and **Insert** in the test editor to select the menu item.
3. In the **User Action Details** section, select an item in the list of object's actions. You can also enter a value for the timeout.

   Result

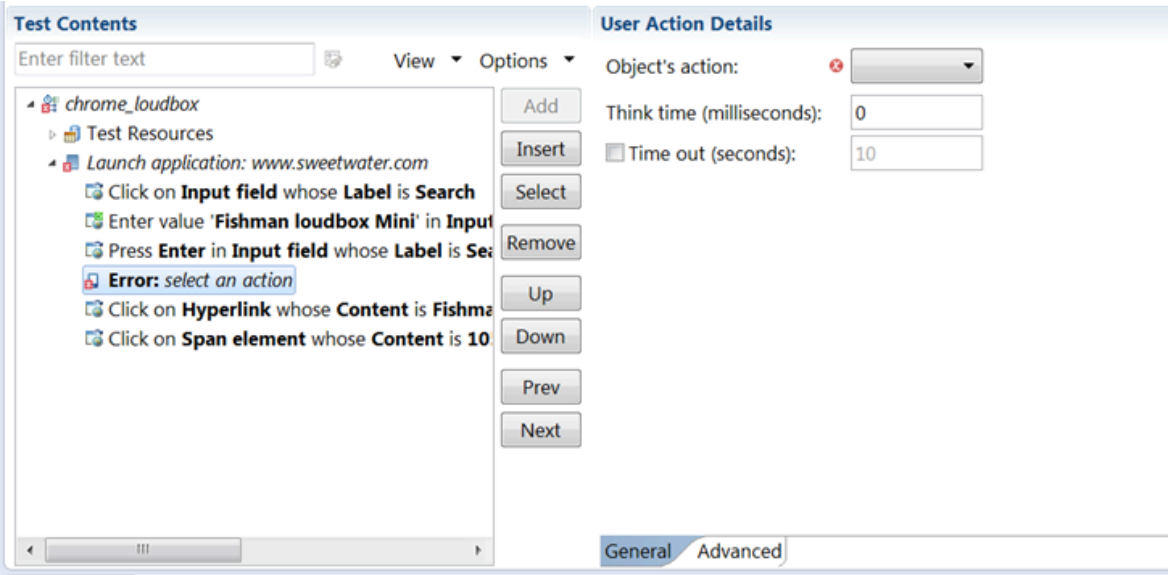
   The new navigation action is inserted before the script item you had initially selected in the launch node.
4. Save the test.

Using the Go To URL action to launch another web application

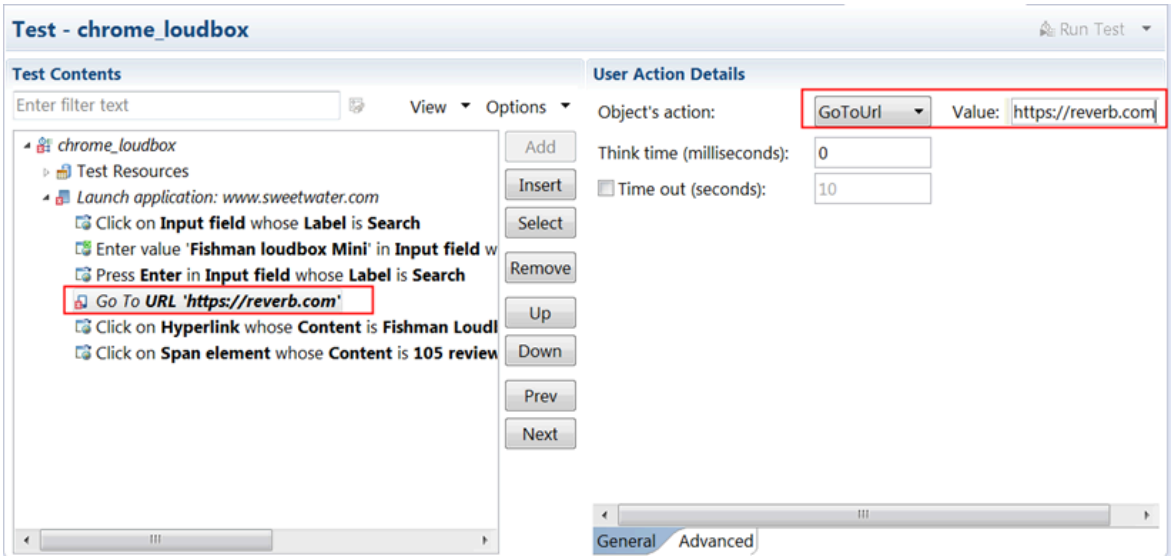
You can use the **Go to URL** action to add a step that launches a new web application. This action inserts a new URL into the address field of a browser window and thus redirects playback to another web application.

1. In Rational® Functional TesterHCL OneTest™ UI, open a Web UI test script, and in the **Test Contents** area, click where the navigation action is to be inserted.
2. Click **Insert** and select **Navigation action (Web)**.

   The new navigation action, which is displayed initially as **Error: select an action**, is inserted before the step you had initially selected.



3. In the **User Action Details** section, select **goToUrl** in the list of Object's actions, and enter the URL for the web application in the **Value** field.



Result

The error in the test is replaced by the user action **Go to URL**. You can also enter a value for the timeout.

4. Optional: Move the new navigation action Up or Down in the test script.
5. Save the test and run it to verify the **Go to URL** action.

Results

After adding the **Go to URL** user action, you can insert additional browser navigation actions to do things such as maximize, refresh, and restore the browser window during playback.

Simulating keyboards and special keys actions on Web and native application windows

When you record a test, you can simulate keyboard or special keys actions on web and native application windows, or navigation dialog boxes on browsers on Windows operating system. To simulate these actions, you can add the navigation action **PressKey** or **InputKeys** to the test script. Starting from Rational® Functional TesterHCL OneTest™ UI V9.2.1.1 of the product, you must enter the values in the correct format.

Before you begin

- You must have created a Web UI test from a recording.
- The test script must be open in the test editor.

Restriction:

- **PressKey** and **InputKey** are not supported on Mac operating system.
- Only basic key combinations such as [CTRL]+[A], [CTRL]+[C], [CTRL]+[V] that are executed in browsers are supported.
- New key combinations are not displayed in test scripts recorded with versions earlier to V9.2.1.1. If you want to benefit from the new key combinations in existing test scripts, you must remove the existing test steps and add new steps.
- Double-byte characters are not supported.
- **PressKey** and **InputKey** and any other actions that use Robot APIs are not supported in the following scenarios:
  - Screen is locked
  - Application being tested is minimized
  - Remote desktop window is minimized

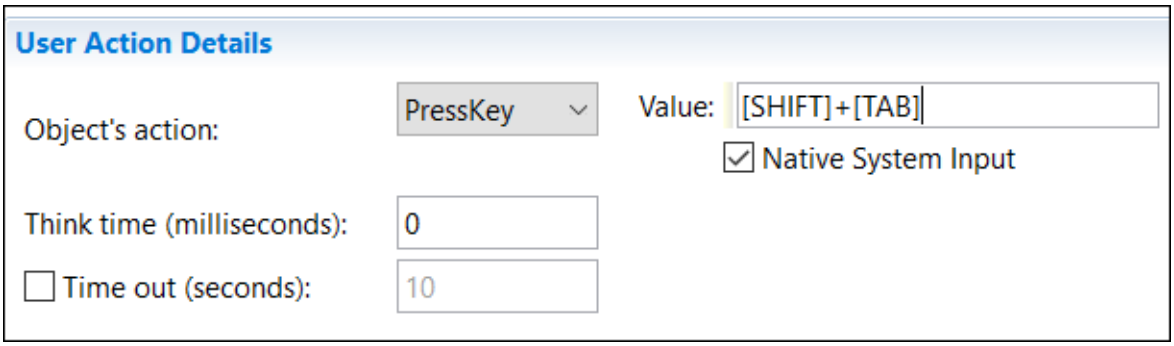
  This restriction is applicable irrespective of whether you initiate the execution from the product, command-line interface, or any integrations that invoke the test.

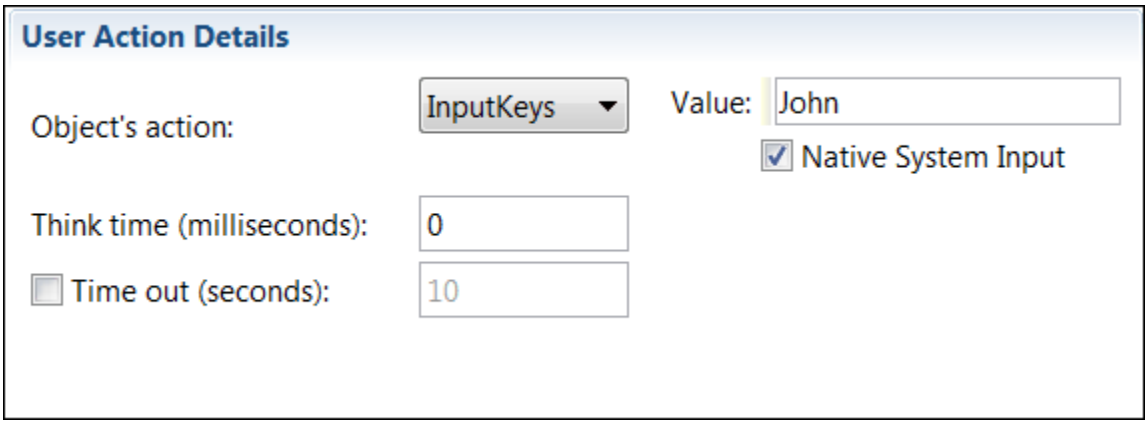
1. In the test script, place the cursor where the step must be added, and click **Insert > Navigation action**.
2. In the **Test Element Details** section, select **InputKeys** or **PressKey** in the list of objects actions.
3. For **InputKeys** and **PressKey** navigation actions, specify the string in the edit field that is automatically entered at the playback time when the step is replayed. The **PressKey** navigation action is used to simulate the **TAB** key or specific kind of keys. The list of supported navigation key values and their formats are as follows:

Key value	Key format
Tab	[TAB]
Shift	[SHIFT]
Enter	[ENTER]

Key value	Key format
Esc	[ESCAPE]
PgUp	[PAGEUP]
PgDn	[PAGEDOWN]
Home	[Home]
End	[End]
Left arrow key	[Left]
Right arrow key	[Right]
Up arrow key	[Up]
Down arrow key	[Down]
F1 to F12	[F1] ….[F12]
Ctrl	[CTRL]
Alt	[ALT]
Delete	[DEL]
"="	[=]
"-"	[-]
"+"	[+]

For example, [CTRL]+[ALT]+[S], [CTRL]+[C], [CTRL]+[V]





4. Optional: Select **Native System Input** to simulate the input keys or special key actions at the screen level on the active window. This option is used to handle non-HTML windows such as file browse dialogs. To interact with file browse dialog boxes, you should keep your machine unlocked and the browser dialog box is in the foreground.

   Notes:
   - This option is not available for AFT parallel executions where multiple browser instances are running.
   - This option is not available on Mozilla Firebox browser navigation.

5. Save and run the test.

Defining a variable to run a test with a selected browser

If you run a test from the command line, as part of a schedule from IBM® Rational® Performance TesterHCL OneTest™ Performance, or from IBM® Rational® Quality Manager, you can define a variable in the test to specify the web browser to use for running the test. The reserved name for the variable is `RTW_WebUI_Browser_Selection`. After defining the variable, if you run the test from test workbench, the browser selected in the **Run configuration** dialog box takes precedence over the browser specified in the variable.

Before you begin

Now, IBM® Rational® Functional TesterHCL OneTest™ UI creates a test variable automatically whenever a new Web UI test is recorded or when an old test is used that does not already have this variable defined.

1. In the Test Navigator, browse to the test and double-click it.

   Result

   The test opens.
2. To create a container for the test variables that you create in a test:

a. Open the test, and in the **Test Contents** area, at the top of the test, click **Test Variables**.

b. Select **Add > Test Variables Containers**.

Result

A container named **Test Variables** is created for the user-defined variables.

c. Select the container to rename it.

Result

The **Test Variables Details** area opens for you to type a new name in the **Name** field.

3. To define a variable in a test:
   a. Select the newly created test variable node.
   b. Click **Insert > Variable Declaration**.
   c. Enter the name of the variable, which is a reserved name for this selection variable:
   `RTW_WebUI_Browser_Selection`.
   d. Click **OK**.
   e. In the **Visible in** section, select **This test only** to restrict data to the current test only. Even if another test has a variable with the same name, that variable will not change. Select **All tests for this user** to share the value of this variable when the test runs in a compound test. For the variable to be shared, both tests must have a variable with the same name and must have this option enabled.
4. Assign a specific value to the variable and initialize the variable:

   a. In Initial Value, ensure that the **Text** option is selected.

   b. Use one of the following variables to specify a browser:

Browser	Variable	Variable short form
Mozilla Firefox	Firefox, Mozilla Firefox	ff
Google Chrome	Chrome, Google Chrome	chrome
Internet Explorer v9 32-bit	Internet Explorer, Microsoft Internet Explorer	ie
Internet Explorer v9 64-bit	Internet Explorer, Microsoft Internet Explorer	ie64
Internet Explorer v10 and v11	Internet Explorer, Microsoft Internet Explorer	ie
Microsoft Edge	Edge, Microsoft Edge	edge
Apple Safari	Safari, Apple Safari	safari

5. Save the test.

What to do next

You can now run the test from the workbench, from the command line, as part of a schedule, or from Rational® Quality Manager.

You can use this test variable in **If** conditions to assign different behavior for different browsers. Doing so allows you to create more robust tests that will run successfully in more than one browser.

Verification points

During the playback of tests, verification points verify that certain action has occurred or they verify the state of a control or an object. When you create a verification point for an action, a control, or an object in a recorded test, you capture information about the action, control, or object to build a baseline for comparison when the test is played back.

When you upgrade an application to a later version, you can use verification points to find out potential defects also. If the expected behavior of verification points do not occur during playback, then Rational® Functional TesterHCL OneTest™ UI displays an error.

The following topics provide more information:

- Creating verification points for objects in tests on page 580
- Creating verification points for alert, confirm, or prompt dialogs on page 583
- Creating verification points from the SmartShot View on page 584

Creating verification points for objects in tests

You can create a verification point in a recorded test for any object such as label, button, or others. You can then set the property of the object to color, font size, opacity, or others, so that Rational® Functional TesterHCL OneTest™ UI verifies the property during the playback.

Before you begin

You must have recorded a mobile, Web UI, or Windows test.

About this task

You can edit a recorded test to create a verification point for an object and set the properties. When you run the test, Rational® Functional TesterHCL OneTest™ UI compares the property of the object that was set in the recorded test with the property of the object during the playback to check whether any changes have occurred in the application, either intentionally or unintentionally. After the test run is completed, you can view the test step for the verification point in the test report. For example, you can create a verification point in the recorded test to check if the **Submit** button is of green in color. When the test is played back, Rational® Functional TesterHCL OneTest™ UI verifies the properties and displays the result.

1. Open the test in Rational® Functional TesterHCL OneTest™ UI.
   The test is displayed in the **Test Contents** panel.
2. Select the test step that contains the details of the object for which you want to create a verification point.
3. Click **Insert**, and then select **Verification point**.
   Alternatively, right-click the selected step, and then click **Insert > Verification point**.

A verification step is created and displayed in the **Test Contents** panel.

4. Select the verification step.
   The **Check Action Details** panel for the step is displayed.
5. Perform the actions for the options that you want to include in the verification point, as provided in the following table:

<table>
<tr><th>Option</th><th>Description</th><th>Action</th></tr>
<tr><td>Graphic object</td><td>The option that lists the objects that you can select for the verification point.

Depending on the graphic object that you select, the corresponding options are displayed in the other UI elements of the Check Action Details panel.</td><td>Perform the following steps to select an object in a test:

a. Click Graphic object.
b. Select an object from the list.</td></tr>
<tr><td>Object identified by</td><td>The identifier of the selected object.</td><td>Perform the following steps to select an identifier for the graphic object:

a. Click Object Identified by.
b. Select an identifier such as Xpath or Id of the object from the list, and then provide the values.</td></tr>
<tr><td>Object location</td><td>The location of the selected object.

For example, the Submit button can be below a checkbox on a page when you recorded the test.</td><td>Select the location of the object from the options provided in the list and then provide the values.

The default value is (automatic detection).</td></tr>
<tr><td>Verify attribute</td><td>The attribute to be specified for the selected object.

Some object attributes are dependent on others. Therefore, when you select an attribute, you must select the values that are related to the selected attribute.

If an object takes longer than the expected time to appear or disappear on the application during the test recording, then you can select the Exist property of the object in the list, and then select the Retry verification point until attribute is verified or time out expires checkbox</td><td>Select the attribute from the list.

Following are two examples that you can perform to set the attributes to an object:

<table><tr><th>Examples</th><th>Tasks to perform</th></tr><tr><td>Example 1</td><td>You can perform the following tasks to set the Exist attribute to the selected object:

a. Select Exist from the Verify attribute list.

b. Select the true checkbox that is next to the Verify attribute list. Rational® Functional TesterHCL OneTest™ UI waits for the object to appear on a</td></tr></table></td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th><th colspan="2">Action</th></tr>
<tr><td></td><td>so that Rational® Functional TesterHCL OneTest™ UI waits until the object appears or disappears on the application during the playback, or continues to retry until the attribute is verified.</td><td colspan="2"><table>
<tr><th>Examples</th><th>Tasks to perform</th></tr>
<tr><td></td><td>page in the application during the playback, and then continues with the test step.

If you do not select the true checkbox, then Rational® Functional TesterHCL OneTest™ UI waits for the object to disappear in the application.</td></tr>
<tr><td>Example 2</td><td>You can perform the following tasks to combine several attributes for the selected object:

a. Select all of from the Verify attribute list.

Two additional lists are displayed where you can select the attributes for the object.

b. Click the add line icon  if you want to view more lists to select multiple attributes for the object.

You can click the remove attribute line button icon  to remove an added attribute if there are more than two attributes.</td></tr>
</table></td></tr>
<tr><td>Time out (seconds)</td><td>The period of time in seconds that Rational® Functional TesterHCL OneTest™ UI must keep trying to verify the attribute of an object.

For example, if the display of an object took 2 seconds to display during the recording, you can set the same period of time as the time-out period or increase the waiting period for the playback.</td><td colspan="2">Optional: Perform the following steps:

a. Select the Time out (seconds) checkbox.
b. Enter the time in seconds that you want Rational® Functional TesterHCL OneTest™ UI to wait to verify the object attribute before the next test step is performed.</td></tr>
<tr><td>Retry verification point until attribute is verified or time out expires</td><td>The option that specifies whether Rational® Functional TesterHCL OneTest™ UI must repeatedly try to verify the attribute of the selected object until it is verified or until the time-out value that is defined</td><td colspan="2">Optional: Select the Retry verification point until attribute is verified or time out expires checkbox.</td></tr>
</table>

Option	Description	Action
	in the **Time out (seconds)** field expires, whichever occurs earlier.	For example, if you selected the color attribute to be verified, you can select this checkbox so that Rational® Functional TesterHCL OneTest™ UI waits until the object is verified before it continues to the next test step.

6. Save the test.

Results

You have successfully created a verification point for an object in the recorded test.

What to do next

You can perform the following tasks:

- Run the test. See Running a Web UI test on page 1243.
- View the unified report for the test. See Unified reports on page 1455.

Creating verification points for alert, confirm, or prompt dialogs

You can create verification points for alert, confirm, or prompt dialogs in a recorded Web UI test by validating text messages in the dialogs. The text messages are validated during the playback. You can also choose to skip the step for dialogs and continue to the next step if such dialogs do not exist.

Before you begin

You must have read and been familiar with recording a Web UI test. See Recording a Web UI test on page 545.

About this task

You can select the **Validate Text** checkbox in the test step of alert, confirm, or prompt dialogs of a recorded Web UI test script. After you select this checkbox and play back the test script, the text message in the dialog is validated before the user action. The results are captured in the unified report.

You can also select the **Skip the step if the control is not found** checkbox for the test step of alert, confirm, or prompt dialogs in a recorded Web UI test script. After you select this checkbox and play back the test script, Rational® Functional TesterHCL OneTest™ UI checks for the dialog and marks the step as **Pass** even if the dialog does not exist. By skipping the step for dialogs, you can manage the optional dialogs that might or might not display depending on user actions. You can view the unified report to read the message that is logged regarding the presence of the dialog.

1. Record a web application with an alert, confirm, or prompt dialog.
2. Create a duplicate of the required test step such as **Accept the Alert**, **Accept the Confirm**, or **Enter text and Accept the Prompt** in the **Test Contents** pane in the generated test.
3. Select the original test step.
4. Enter the text to be verified in the **Text** box in the **User Action Details** pane.
5. Select the **Validate Text** checkbox.

   Depending on the selected dialog type, the test step is modified to display one of the following statements:

- **Verify that Alert text contains** `dialog text`
- **Verify that Confirm text contains** `dialog text`
- **Verify that Prompt text contains** `dialog text`

This statement ensures that the text message is validated before the user interface action is played back in the dialog.

6. **Optional:** Enter the **Think time** value in milliseconds.
   Think time is a statistical emulation of the amount of time that actual users spend to read or think before they perform an action.
7. **Optional:** Select the **Time out (seconds)** checkbox, and then enter a value for the timeout for the selected step.

   Note: The timeout value specifies the amount of time, in seconds, that the Rational® Functional TesterHCL OneTest™ UI waits for data. If no data is received for the specified amount of time, the session is disconnected.

8. **Optional:** Select the **Skip the step if the control is not found** checkbox to mark the step as **Pass** even if the dialog does not exist.
9. Save the test.

What to do next

You can run the test and verify the results in the unified report. See Running a Web UI test on page 1243 and Unified reports for Web UI, mobile, and Windows tests on page 1456.

Creating verification points from the SmartShot View

You can create some verification points in a test with simplified scripts by using the **SmartShot View**. A verification point can be added for an object or created for the properties of the object.

Before you begin

To be able to create verification points, you must create variables in the test editor. See Declaring and assigning test variables.

About this task

You can create verification points for any of the widgets or widget properties.

To create verification points:

1. In Rational® Functional TesterHCL OneTest™ UI, open the test script, and in the **Test Contents** area, click an action item for which you want to create a verification point.
2. In the **SmartShot View**, select an object in the **SmartShot** tab, an item in the hierarchical list of **Elements**, or a property in the table.

3. Right-click and click **Create Verification Point for propertyName**. Note that the properties displayed will be limited to those available for the selected object. A new step is added to the test script for the verification point.
4. Select a value in **Graphic object** and a value for the objects property in **Verify attribute**.

You can also modify a verification point using the drag and drop method:

5. Select a step with a verification point. In the **SmartShot** tab, select a property available in the table of properties for the target object, drag the property and drop it on the test script or on the verify attribute field. The object property and its value are showing in the test step and in the appropriate fields in the **Check Action Details** area.
   You can also add images as main property in a verification point with the same method. You select an image in the **SmartShot** view, drag and drop it on the **Identified by** field.
6. Save the test.

Assigning a test variable to an object property

You can use the value of a UI object property and assign the value to a test variable in all the UI tests created for Android, iOS, hybrid, or native mobile applications, desktop applications, and web applications. The process of assigning a variable to an object property is called a variable assignment.

Before you begin

To create a variable assignment, you must have declared a variable. See Declaring and assigning test variables.

About this task

You can extract a substring from the content of an object property by using a regular expression and then assign the substring to a variable. For example, you can extract *User Documentation* from the Content property of a hyperlink, and then assign the string to the *trimVar* variable. Further, you can include *trimVar* in the subsequent steps and execute the test. You can view the variable value *User Documentation* in the unified report.

You can also use variable assignment to check if an object exists and return *true* or *false* to the variable. Further, based on the value of the variable, you can choose to branch the test steps by using an `If-Else` block. For example, you can assign the **check_state** variable to the **Object's property > Exist** property of the **Login** button, and then use conditional statements to either log in or not.

You can create variable assignments in all the UI tests that are created for Android, iOS, hybrid or native mobile applications, desktop applications, and web applications.

Note: When you run a test from the mobile client on mobile devices, it uses the same values that you used during recording. If you modify the test script and create a dataset or variable, or if you add a condition, a loop, custom code, references or add other statements, they are not taken into account by the mobile client at

run time. To verify that the initial recorded values are substituted with variable data, you must initiate the test run from Rational® Functional TesterHCL OneTest™ UI.

1. Open the test, and in the **Test Contents** area, select a test element.
2. Select **Insert > Variable Assignment**.
   The **Test Editor** window opens and lists the variables available to the test.
3. Select the variable for which you want to assign a value, and then click **OK**.
   A variable assignment statement is inserted before the selected element.
4. Select **UI object property** in the **Set to** box in the **Variable Assignment Details** area.
5. Select the values for the following fields:

Option	Description	Action
Graphic object	This option lists the objects that you can select for variable assignment. Depending on the graphic object that you select, the corresponding options are displayed in the other UI elements of the **Variable Assignment Details** panel.	Select an object from the list.
Object identified by	This option lists the identifiers of the selected object.	Select an identifier such as Xpath or Id of the object from the list.
Object location	This option lists the possible locations of the selected object. For example, the **Login** button can be below a checkbox on a page when you recorded the test.	Select the location of the object from the options provided in the list and then provide the values. The default value is **(automatic detection)**.
Object's property	This option lists the properties of the selected object that must be used for variable assignment.	Select one of the following options from the list: ◦ Select a property such as Title, Content, or Xpath of the object that must be used to extract the value using a regular expression. Go to step 6 on page 587. ◦ Select **Exist** to verify whether the selected object exists and returns the value to the variable. Go to step 7 on page 587.

The values of the properties displayed in the lists are different for web, Android and iOS apps.

6. Enter a regular expression to extract a substring from the value of the object property in **Extract value using Regex**.
   For example,

   ```
   User(.*)
   ```

7. Save the test.
   A set statement is added to the test with the value you chose.

   For example, the following set statement is added if you selected **Extract value using Regex**:

   ```
   Variable 'trimVar' is set to 'Extract value using regex User(.*)' from property 'Content'
    of Hyperlink whose Content is User Documentation located at position 1
   ```

   If you selected **Exist**, the following set statement is added:

   ```
   Variable 'trimVar' is set from property 'Exist' of Heading H4 whose Content is User Documentation
   ```

Adding a loop

To run a test or part of a test repeatedly for a specified number of times or duration, add a loop to the test. You can add a loop to the Launch Application node or In Application node.

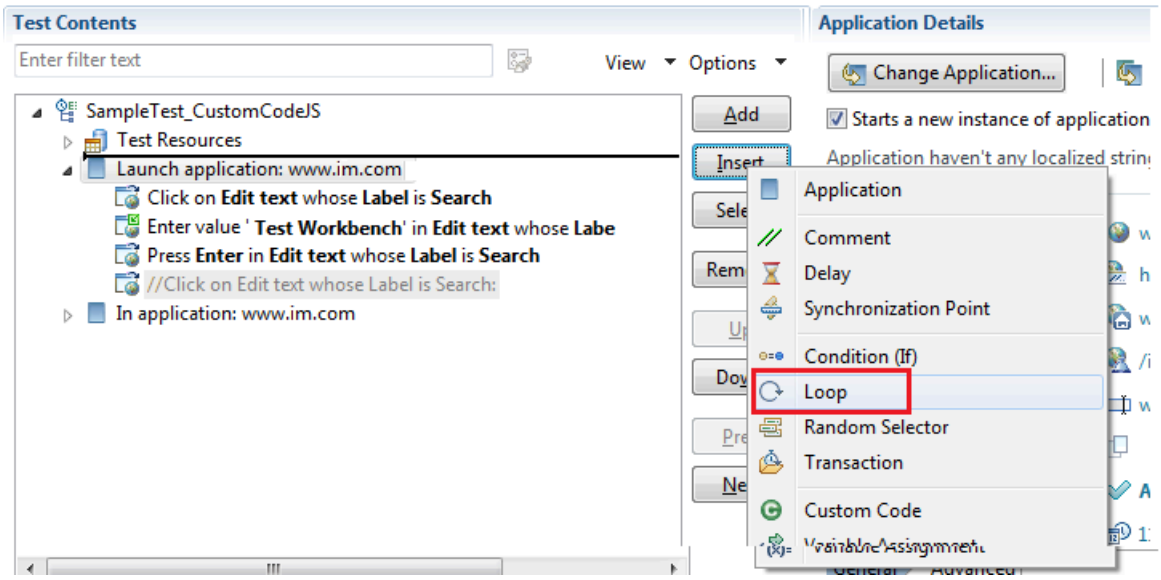
About this task

When you add a loop to specific steps in a test, those steps are split to create a new In Application node. For information about splitting a test, see Splitting test actions on page 592.

When you associate a dataset to a test and want the test to fetch data from all of the rows of a dataset, add a loop to the test and set the loop to run infinite. You can also set the loop to run for a specific duration or number of times. For information about associating a dataset to a test, see Associating a dataset with the test.

To add a loop to the Launch App or In App nodes:

1. In the Test editor, click the Launch Application or In Application node.
2. Click **Insert > Loop**.



In the confirmation dialog box, click **Yes**.

3. In the Loop Details area, specify a name for the loop.
4. In the **Loop Details** area, type the number of iterations for the loop to repeat.

Option	Description
Count-based	Runs for the number of iterations that you select.
Time-based	Runs for at least the time that you specify. The loop always finishes the iteration. For example, if you select a time of 1 second and a loop takes 10 seconds to run, the loop finishes one iteration, and then checks the time.
Infinite	Runs until the test stops.

Note: If a test includes multiple loops with dataset values and a new dataset value is required for the first iteration of the second loop, then a dataset increment is required before the second loop runs. To do this, you must insert a data source controller by clicking **Insert > Data Source Controller** before the second loop starts and then select the required dataset. You can then select the **Increment** option for the data source from the **Data Source Controller Details** pane that triggers the retrieval of the dataset value to automatically choose the new dataset value.

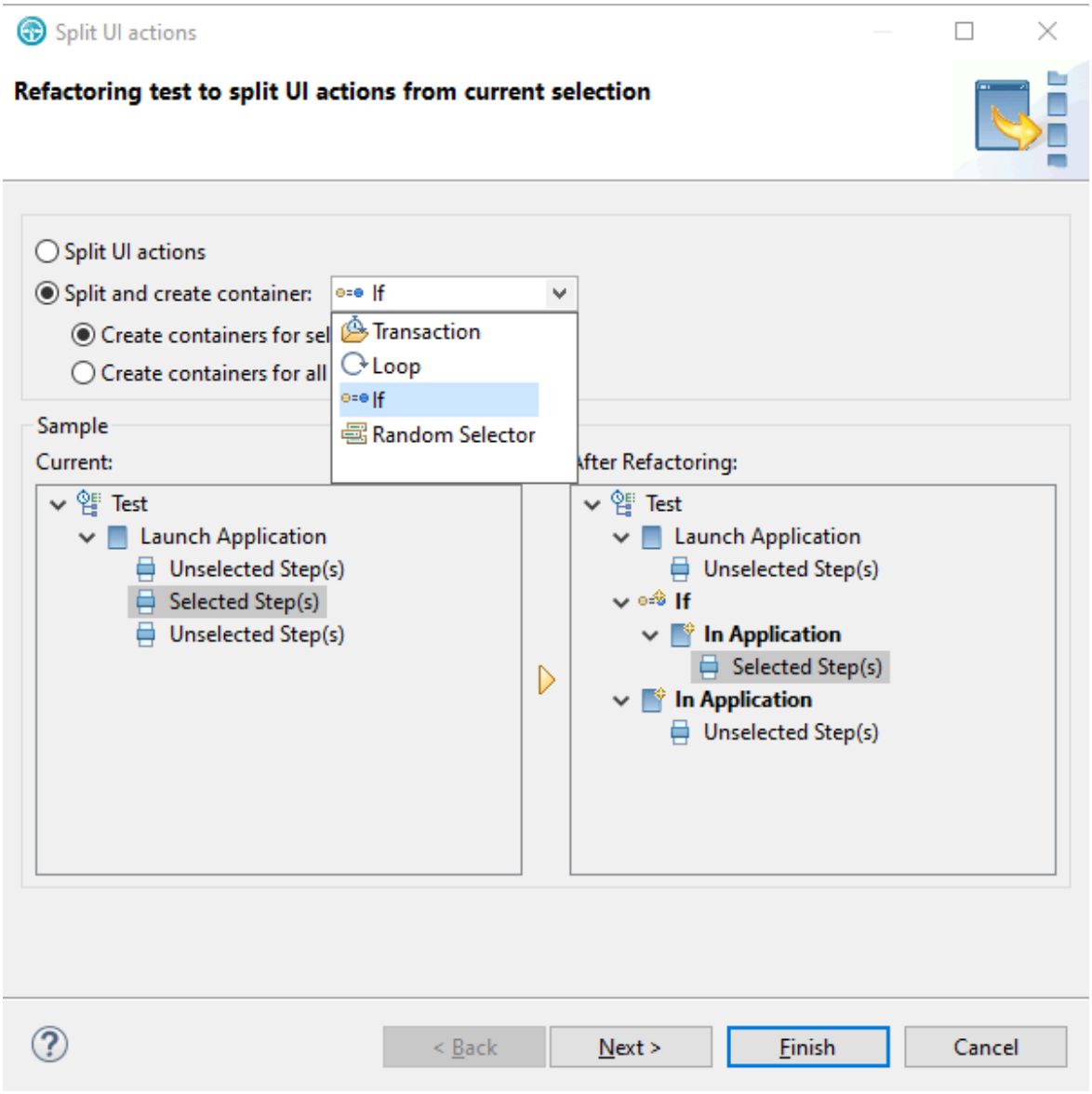
5. Optional: Select **Control the rate of iterations**, and type your preferences for the pacing rate.
   In specifying a number of iterations for a unit of time, you set a fixed period for the iterations to complete. If you select **Randomly vary the delay between iterations**, the total delay is randomly distributed. If you clear this check box, the same delay occurs between each iteration.

Note: Statistically, the **Randomly vary the delay between iterations** option sets delay amounts at random from a negative exponential distribution with the same mean as the fixed delay value. The negative exponential distribution has a long "tail," which means that a very small number of delays will have very large values. Therefore, make sure that the application you are testing is not negatively affected by long periods of inactivity (such as a timeout that disconnects the user).

Adding conditional logic to tests

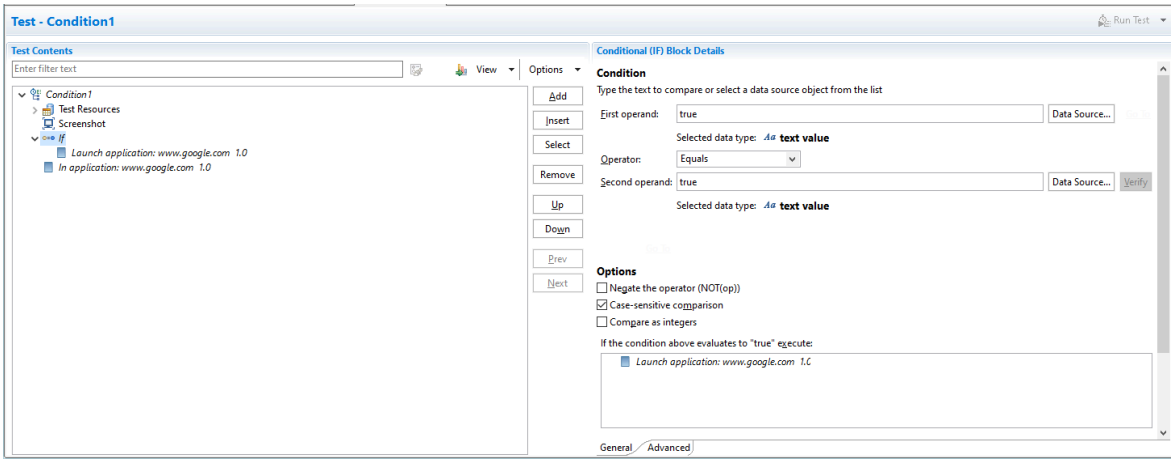
You can insert the `If-Else` logic for a section of test to make those steps run only if a specific condition is met.

1. Double-click a test in the **Test Navigator** pane. The test opens.
2. Select the steps to be added to the `If` block, right-click and then click **Split Mobile or Web UI actions**.
3. Select **Split and create container**, and then select **If** from the drop-down list.
4. Click **Finish**.



Result

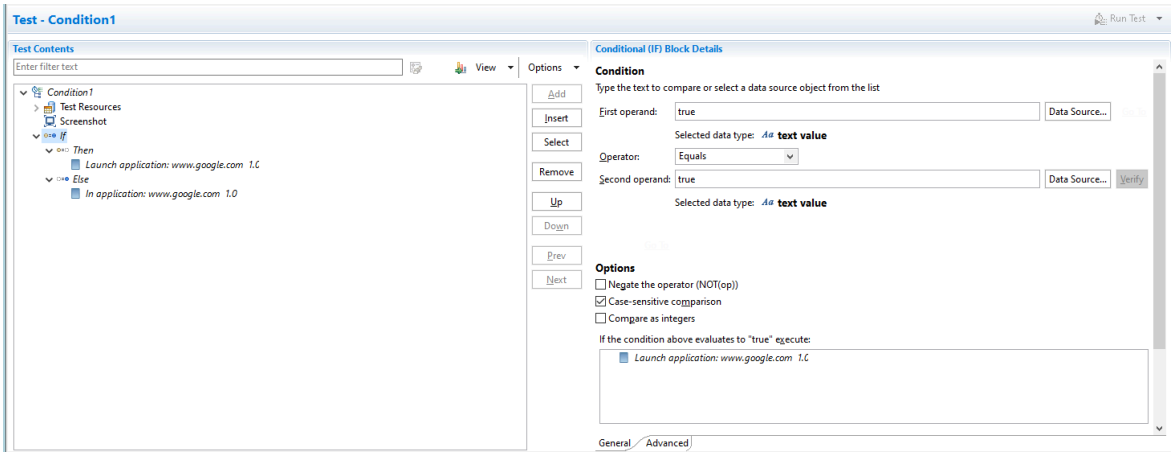
The selected steps along with the Launch application node are moved to the `If` block and the rest of the steps are added to the In application node.



5. Click the `If` block and then click **Add > Condition (If) - ELSE Block** to add the `Else` block.

   Result

   The `Else` block is created and the steps from the `If` block moves to `Then` node.



6. Add the following conditions in the **Conditional (IF) Block Details** area, under:
   - Next to the **First operand** field, click **Data Source**, and then select a data source to be compared with the string in the **Second operand** field, or type a value in the **First operand** field.
   - In the **Operator** field, indicate the basis of comparison of the two operands. Note that the two operands are strings.
   - Next to the **Second operand** field, click **Data Source**, and select a data source to be compared with the **First operand**, or type a value in the **Second operand** field. When the defaults are used (both operand fields set to true and the **Operator** field set to Equals), the block is always processed.
7. Under **Options**, choose any of the comparison types by selecting the checkbox:

Option	Description
Negate the operator (NOT(op))	Negates a condition when you perform a comparison. You can use the **Negate the operator** option to obtain an opposite result of any test condition. For example, if the result of any condition

Option	Description
	is `true`, the result returns to `false`, and if the result of any condition is `false`, the result returns to `true`.
Case-sensitive comparison	Performs a case-sensitive string comparison. You can use `Equals` in the **Operator** to perform a case-sensitive comparison in a test.
Compare as integers	Performs a digit comparison when the two operands are integers to return the result, `true` or `false`.

8. Save the test.

Splitting UI actions

After you record a test, you can split the test actions into multiple test segments with different nodes. With the test-splitting capability, you can record a relatively long scenario with many functional steps against an application and then, in the editor, modify the target apps. Then you can generate multiple tests from a single recording that you can replay in a different order with a schedule.

To split UI actions:

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. In the test editor, select one or more actions in the test script for splitting into one or more application nodes. You can select elements, except for variable containers, that are immediate children of the root node of the test.
3. Right-click the selected elements, and then select **Split UI actions**.
4. In the refactoring test dialog box that opens, examine the changes to be performed as a result of the split. You can leave or clear the options if you do not want certain data to be correlated.
5. Click **OK**.
   Result
   One or more app nodes *In application: AppName* are created in the test script from the selected test element.
6. Optionally: you can change the target app to be tested for a selected app node. To do so, select an app node, click the **Change application** button and in the list of applications available, select a new app. Then select the **Starts a new instance of application selected below**. To apply the change of app to the all the test nodes, that is to the whole test suite, click the  .
   Result
   The test nodes turns from *In application: AppName* to *Launch application: AppName*.
7. Save the test.

Optimizing the playback time of a test

When you play back a Web UI test, by default, the test collects a lot of data. This data includes screen shots, response time breakdown data, and highlighted UI controls. The size of a test depends upon the data it contains; playback time increases as the size of a test increases. To optimize the playback time, you can choose to collect only the data which you need.

About this task

For example, when you play back a test, screenshots of all of the steps are captured and shown in the UI Test report. Most of the times, you might require screenshots only for the failed steps.

Note: For the Safari browser, you can capture only the web page screenshots.

Also, if the computer on which you are playing back the test is locked or you are working on other applications on that computer, the captured screenshots will either be dark black or belong to applications that are not relevant to the test. When you capture only page screenshots, you can lock the screen or work on other applications on the computer.

Similarly, you might be interested only in testing the functionality of the application, and the performance testing might be covered by someone else. You can choose not to capture the response time breakdown data thereby improving the speed of the playback of your test.

You can apply playback optimization options for the test or for the workbench (in Preferences). When you set the optimization options for the workbench, only the desktop screenshots are captured and applied to all of the tests. When you import a test for playback, the configurations on your workbench are used by the test. When you set the optimization options for the test, you can choose to capture screenshots for pages or desktop. To apply playback configurations at the test level, use the pre-defined variables and values mentioned in the table.

Table with variables for playback optimization:

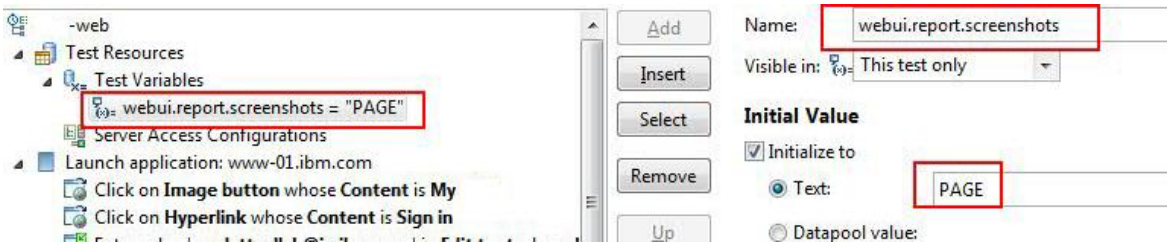
Table 6. Variables for Playback Optimization

Variables	Values	Description
webui.report.screenshots **Note:** If this variable is not set, desktop screenshots for all of	DISABLE	Stop capturing screenshots for all the steps
	PAGE*	Capture the screenshots of pages for all of the steps

Table 6. Variables for Playback Optimization

(continued)

Variables	Values	Description
the steps are captured.	PAGE_ONFAIL	Capture the screenshots of pages for only the failed steps
	PAGE_ONVP	Capture the screenshots of pages for all of the verification points
	PAGE_ONVPFAIL	Capture the screenshots of pages for only the failed verification points
	PAGE_ONFAIL_AND_VP	Capture the screenshots of pages for only the failed steps and all of the verification points
	DESKTOP	Capture the screenshots of desktop for all of the steps
	DESKTOP_ONFAIL	Capture the screenshots of desktop for only the failed steps.
	DESKTOP_ONVP	Capture the screenshots of desktop for all of the verification points
	DESKTOP_ONVPFAIL	Capture the screenshots of desktop for only the failed verification points
	DESKTOP_ONFAIL_AND_VP	Capture the screenshots of desktop for only the failed steps and all of the verification points
webui.responsetime	False	Do not collect response time
webui.highlight	False	Do not highlight UI controls during playback



* If you specify the PAGE value, screenshots are captured differently on different web browsers and take more time:

- Google Chrome
  - ▪ The portion of the web page that is hidden under the scroll bar is not captured.
  - ▪ The browser is in focus even when minimized.
- Internet Explorer
  - The portion of the web page that is hidden under the scroll bar is captured but in black.
  - The browser window resizes often while the screenshot is taken.

To apply playback optimization options for the workbench:

1. In the workbench, click **Window > Preferences > Test > Test Execution > UI Test Playback**.
2. Click the **Report** tab and select the following options as required:
   - To capture the screenshots of only the web browser page that runs the application under test, select the **Capture browser page screenshots** check box.
   - To capture screenshots, click the **Take screenshots while playing back** check box is selected, and select one of the options to take screenshot for.
   - To highlight the recorded page elements during playback, select the **Highlight the page element** check box.
3. To collect performance data, click the **Desktop** tab and select the **Collect performance data** check box. If the check box is not selected, the response time breakdown (On App/Off App) data is not collected and the related report is not generated.
4. Click **OK**.

What to do next

You can now play back the tests.

Related information

Running a Web UI test on page 1243

Optimizing Web UI load testing for scalability

Scalability denotes the number of virtual users that can be emulated on an agent machine to generate load. In a Web UI load test, each virtual user requires one instance of a web browser to be run. Because each instance of a browser consumes key computer resources such as memory, CPU and Network data, you must tune a few of the parameters to comfortably run appropriate number of browser (virtual users) on an agent machine.

About this task

Here is the sample configuration of a computer that was used to determine the memory consumption:

Table 7. Sample configuration

Operating System	Red Hat Enterprise Linux 6.6
System main memory	4 GB
Free memory before test run	~3 GB
Firefox (empty) memory consumption	130 MB
Firefox (medium-size app) main memory	200 MB

With this configuration, about 15-20 (300 MB / 200 MB) browser instances or virtual users can be emulated on on agent machine. Note, the browser memory consumption may vary based on the dom size of the web applications.

Tuning parameters

By default, the playback process of the Web UI test creates a thread pool of size 10 threads. This behavior affects the execution time when more than 10 virtual users per agent machine are emulated. This limit can be improved by passing the `=-DrptDynamicThreads` parameter to the agent machines.

Also, the Web UI test playback is a java process with maximum memory limit set to 512MB through the `-Xmx` argument. This parameter might yield more memory for browser instantiations.

To optimize Web UI load test:

1. In the Test Navigator view, double-click the location assets for optimization.
2. In the **General Properties** tab, double-click `RPT_VMARGS` row.
   If `RPT_VMARGS` parameter is not available, click **Add**.
3. In **Property Name**, specify `RPT_VMARGS` and in **Property Value** specify `-DrptDynamicThreads -Xmx512m`.
4. Click **OK** and save the change.

Actions from the SmartShot View

The **SmartShot View** displays the screen captures that were captured during the recording of the application under test. Use this view to display and select user interface (UI) elements and optionally add verification points to the test script.

Adding user actions in a test from the SmartShot View

The **SmartShot View** offers graphical and hierarchical views of the current step in a test. The view displays a table of properties associated with a selected object. You can also use this view to quickly create user actions, adding steps to the test using the selected graphical elements.

Before you begin

You must have created a test from a recording and have the test script open in the test editor.

About this task

You can add actions for any of the widgets in the **SmartShot** view.

1. Click an action item in the **Test Contents** area of the test editor.
2. Select a graphical object or the corresponding element in the hierarchical list **Elements** in the **SmartShot** view, and then right-click and select **Add user action for this element**.
   Result
   In the test editor, a step is added just before the current selected node.
3. Select a value for the action of the object in the **User Action Details** section. An easier method is to drag an object in the **SmartShot** view, and drop it in the test script.
4. Save the test.

Modifying a step in a test from the SmartShot View

You can modify the step targets in a test from the **SmartShot View**. Two options are available from the menu items. One is used to modify an action in a test script and assign a new object as target of the action. The other option is used to define execution variables for the selected object and give a new value to the property associated with the object.

Before you begin

You must have created a test from a recording and have the test script open in the test editor. In the Test Contents area you must have selected the action item for which you want to modify the step.



Note: When you run a test from the mobile client on mobile devices, it uses the same values that you used during recording. If you modify the test script and create a dataset or variable, or if you add a condition, a loop, custom code, references or add other statements, they are not taken into account by the mobile client at run time. To verify that the initial recorded values are substituted with variable data, you must initiate the test run from Rational® Functional TesterHCL OneTest™ UI.

About this task

You can modify a step target in a test by either assigning an object as the step target or by creating a variable assignment from a propertyName.

To assign a new object as step target:

1. In the **SmartShot** view, select a graphical object or the corresponding element in the hierarchical list **Elements**, and then right-click and select **Use this element as step target**.
   Result
   In the test editor, the current step target is replaced by the selected graphical object.
2. In the **Test Element Details** section, you can change the action or location initially specified to fit the new target.

To assign a variable and set a new value for the objects property:

3. Select an object in the **SmartShot** or **Elements** view, and then right-click and select **Create variable assignment from propertyName.**
4. In the dialog box, search for a variable that was created in your test. To do so, enter a name to filter the list of available variables and click the one matching the name. Click **OK**.
5. Save the test.

Adding the drag-and-drop action in a Web UI test

When you record a Web UI test that requires you to drag and drop an object manually onto another object, you must edit the recorded test to add the drag-and-drop action.

Before you begin

You must have recorded a Web UI test. See Recording a Web UI test on page 545.

About this task

When you record a Web UI test that requires you to drag an object and drop it in a different location on the same web page, the drag-and-drop action provides an input to complete the task. For example, you can drag an amount for payment among the available denominations, such as 1000, 3000, and 5000 and drop it on the payment field. When you drag '1000' to the payment field, Rational® Functional TesterHCL OneTest™ UI captures the draggable object '1000', but the drag-and-drop action is not evident. You must then edit the test to add the Drag-drop action and specify the Xpath, Id, or the x, y coordinates that identifies the drop location for the object. The subsequent playbacks of the test display the added action.

1. Double-click the test to open it from the **Test Navigator** view.
2. Select the test step that contains the object for which you want to add the drag-and-drop action in the **Test Contents** area.
3. Select **Drag-drop** from the **Object's actions** list in the **User Action Details** section.
   The **Drop to object** list with a text box is displayed.
4. Select one of the following options to specify the drop location for the object:

Option	Action
To Point x, y	Enter the x, y coordinates of the drop location in the text box.
By Id	a. Click the required object in the **SmartShot View**, and then copy the Id. b. Paste the Id in the text box.
By Xpath	a. Click the required object in the **SmartShot View**, and then copy the Xpath from the **Properties** panel. b. Paste the Xpath in the text box.

   Result

   The step is modified to display a Drag-drop statement with the value you chose. For example, `Drag-drop Division to target object with Xpath //*[@id="payment"] whose Content is 1000.`
5. Save the test.

What to do next

You can run the test and verify the results of the drag-and-drop action in the unified report. See Running a Web UI test on page 1243 and Unified reports for Web UI, mobile, and Windows tests on page 1456.

Enhancement of test script robustness

Web UI tests in IBM® Rational® Functional TesterHCL OneTest™ UI might encounter issues with object recognition during playback, that can lead to test failures or errors. To address the issues and to improve test robustness, you can use various methods.

One common reason for step failure is when the version of an application changes. For example, if a test was recorded with an earlier version of an application and you playback the test in a newer version of the application that has updated objects or object locations, Rational® Functional TesterHCL OneTest™ UI might fail to recognize the new objects during the playback. Another factor that contributes to step failures is the variation in data between the time of recording and playback. For example, the date and time when the test was recorded can be different from the date and time during the test playback.

To enhance the robustness of your test scripts, you can consider to implement the following methods:

- **Modifying object identification**
  - You can modify and select a more suitable property and value for the target objects within test steps so that Rational® Functional TesterHCL OneTest™ UI can identify the objects easily during the test playback. See Object properties on page 600.
  - You can use images as the main property for step targets or modify the generated images in a test so that Rational® Functional TesterHCL OneTest™ UI can identify images easily during the test playback. See Image recognition in a test on page 610.
  - You can use regular expressions to replace recognition properties in verification points. See Assigning a test variable to an object property on page 585.
- **Adding object locator conditions**

  You can include object locator conditions within test steps so that Rational® Functional TesterHCL OneTest™ UI can locate the target objects easily during the test playback. See Object location in a test on page 605.
- **Applying Responsive Design conditions**

  You can include the Responsive Design conditions for actions in tests that should be executed only once during the test playback. See Responsive design condition on page 611.
- **Performing asynchronous actions**

  You can select the **Perform asynchronously** option in a test for certain actions of objects such as *click*, *hover*, *press Enter*, and so on so that the actions are performed asynchronously during the test playback. See Actions performed asynchronously on page 611.

- **Using regular expressions for dynamic elements**

  You can use regular expressions to identify dynamic titles and URLs of multiple windows. See Identification of dynamic window titles on page 612.

By using these methods, you can enhance the robustness of your test scripts.

Object properties

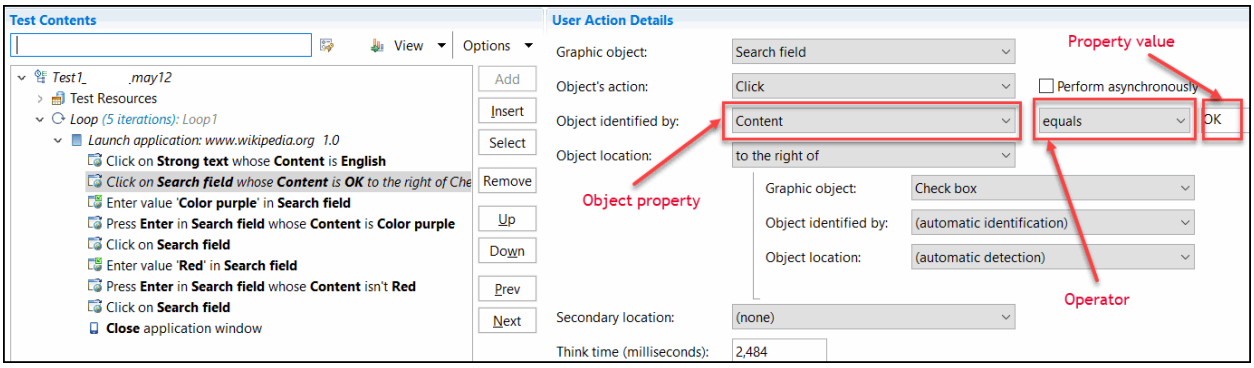
When you record Web UI tests on an application under test (AUT), IBM® Rational® Functional TesterHCL OneTest™ UI captures the properties of objects in the AUT. Capturing accurate object properties is crucial for a successful test playback. Object properties are important to locate and interact with objects in the application during the test playback.

Object properties are automatically captured during test recording and are displayed in the read-only mode within the **Properties** table in the **UI Test** view. Also, when you select a step in the recorded test, the **User Action Details** panel displays the object properties that are associated with the action performed. During the test playback, Rational® Functional TesterHCL OneTest™ UI compares the captured properties and related actions of objects in the test with the objects that are displayed in the AUT, and then tries to identify the objects.

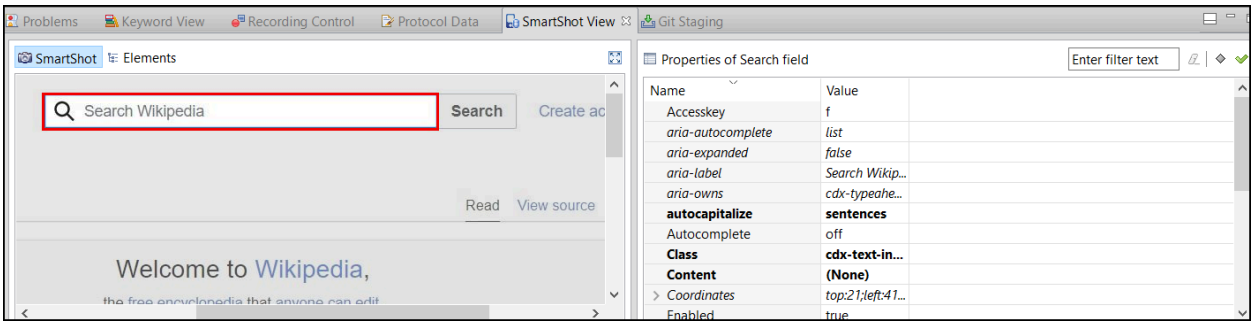
Note: The object properties are different for Android, iOS, or Web UI applications.

For Web UI tests, the **Object Identified by** option in the **User Action Details** panel is comprised of three parameters: object property, operator, and property value. The common object identifiers include **Content**, **Class**, **Id**, Xpath on page 601. You can modify these parameters, or other properties that are displayed in the User Action Details panel to identify the objects during the playback. See Modifying the property used to identify an object in a test script on page 601.

The following screen capture displays a selected step with the corresponding object property, operator, and property value:



The following screen capture displays the SmartShot view of the selected step and the corresponding **Properties** table:



However, some AUTs use custom attributes to describe properties for UI controls. You can add the custom attributes to a list of preferred identifiers and prioritize them so that they are recognized as the main object identifiers in the application under test. See Specifying the properties used for UI controls on page 603.

Object properties play an important role in Rational® Functional TesterHCL OneTest™ UI to accurately identify and interact with objects during the test playback. By understanding how to modify and enhance these properties, you can optimize your test scripts and ensure accurate object identification.

Modifying the property used to identify an object in a test script

When a test is recorded, the property that is used in the test to identify a graphic object might be inaccurate and cause the step to fail during the playback. You can modify an object property and its value, as well as the operator, to improve test robustness.

About this task

You can change the value of the default identification properties for the UI controls.

Note: For the XPath default identification property that is used for UI controls, its value is calculated automatically when recording the test, and all operators are compatible with this identification property. XPath property can be modified but with some limitations: if the XPath value is modified with a value that is not captured when recording the test, the Screen capture is not highlighted in the Mobile Data View and only 'equals' operator is accepted, no other values or regular expressions can be used.

You can also replace a generated text property with an image property to identify a target object. For details about object recognition in mobile and Web UI tests, see Enhancement of test script robustness on page 599.

To modify an object property, you can use one of the following methods: drag and drop, copy and paste, or the context menu.

1. Click a step in the script. The object captured during the recording of this test step is highlighted in the **SmartShot** view.
2. In the Properties table of the **SmartShot** view, select a relevant property. Click the **View main property only** icon in the filter tool bar to see the main properties displayed in bold, or click the **View verifiable property only** icon.

3. Drag the property to the **Object Identified by**field in the test editor. Or, right-click the property in the table and select **Copy**, then **Paste** in the **Object Identified by** field. Another method is to right click a property in the table, and select **Identify step target using property**.

   A menu item is available for each candidate property in the context menu of the **SmartShot** view (see Figure 2). If the newly selected property is inappropriate, a message warns you that object recognition may be broken. The property name and its value are replaced in the **Object identified by** fields.

   When the test is recorded, the default operator for identifying an object in a test run is equals but there are many other operators that you can use to identify objects in a mobile or web UI application. They can be used for verification points in a test to verify an attribute, for example. See Creating verification points from the SmartShot View on page 584 and Creating verification points for objects in tests on page 580.

   Figure 1. Modifying the object property and value from the Properties context menu

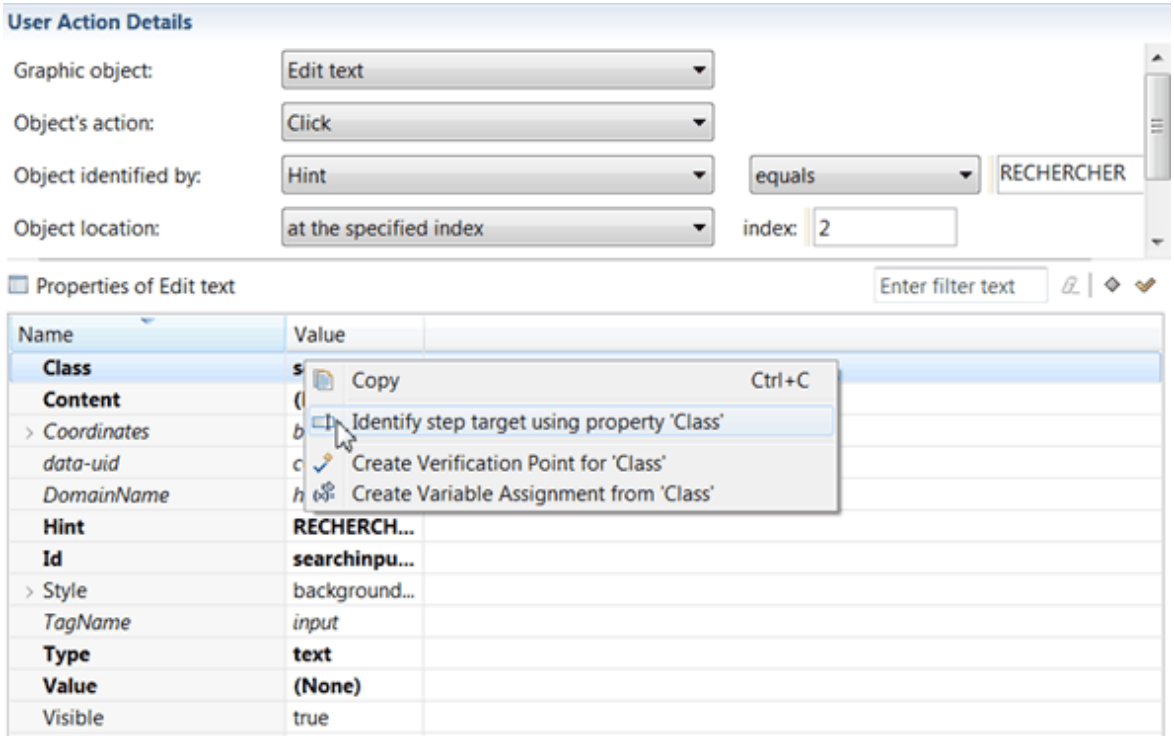
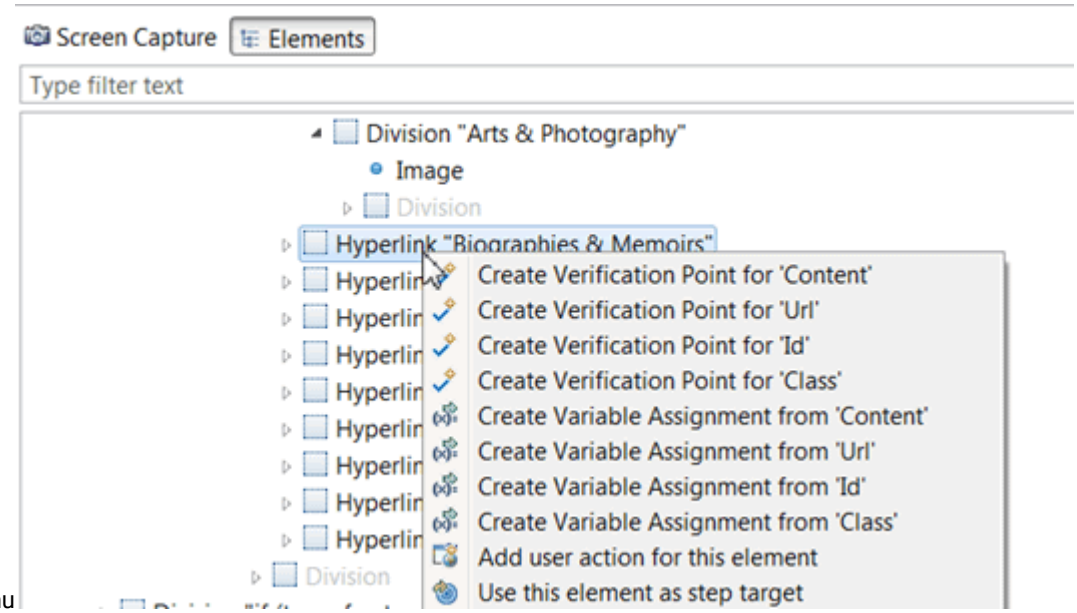


Figure 2.Modifying the property and value of an object captured during the test recording with a candidate property from the SmartShot context menu



4. Save and run the test to verify that the object is identified.

Related information

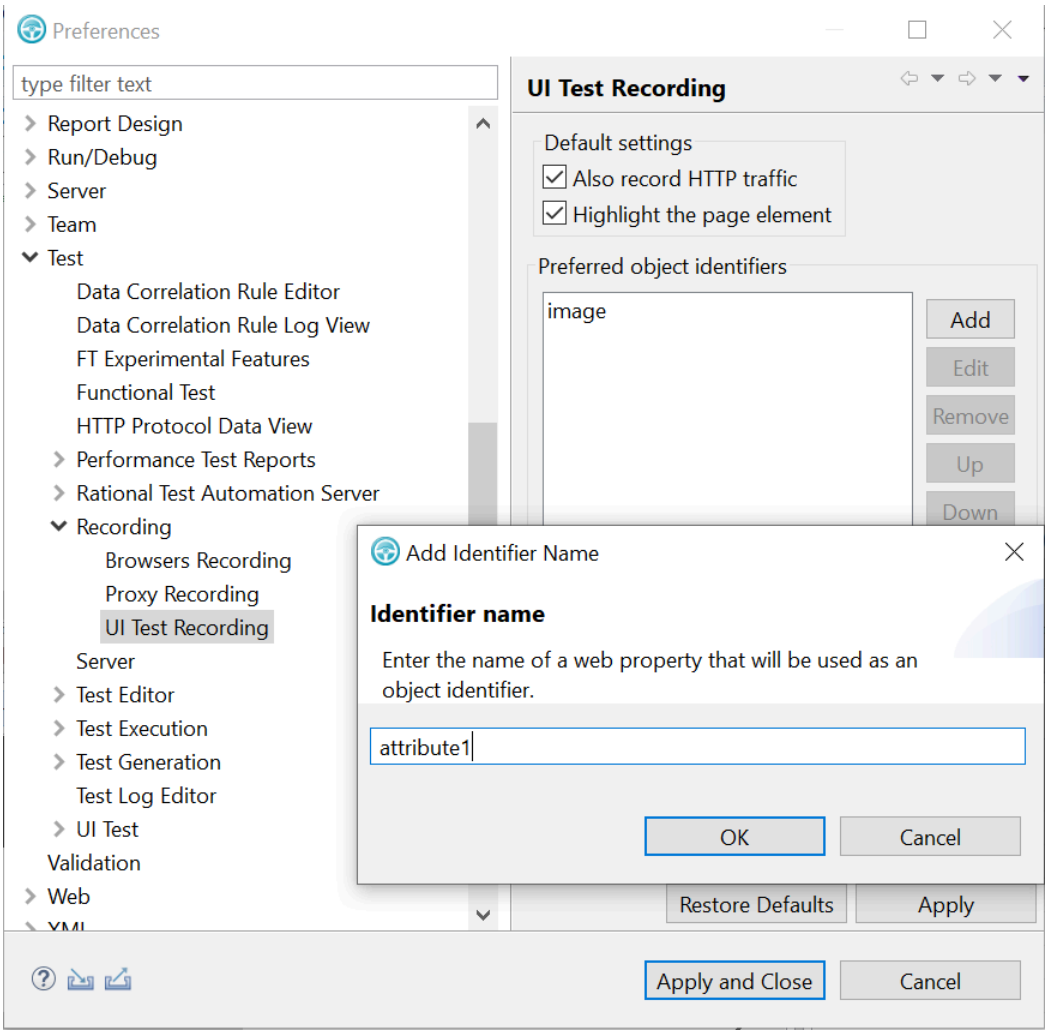
Modifying a step target using an image as the main property

Specifying the properties used for UI controls

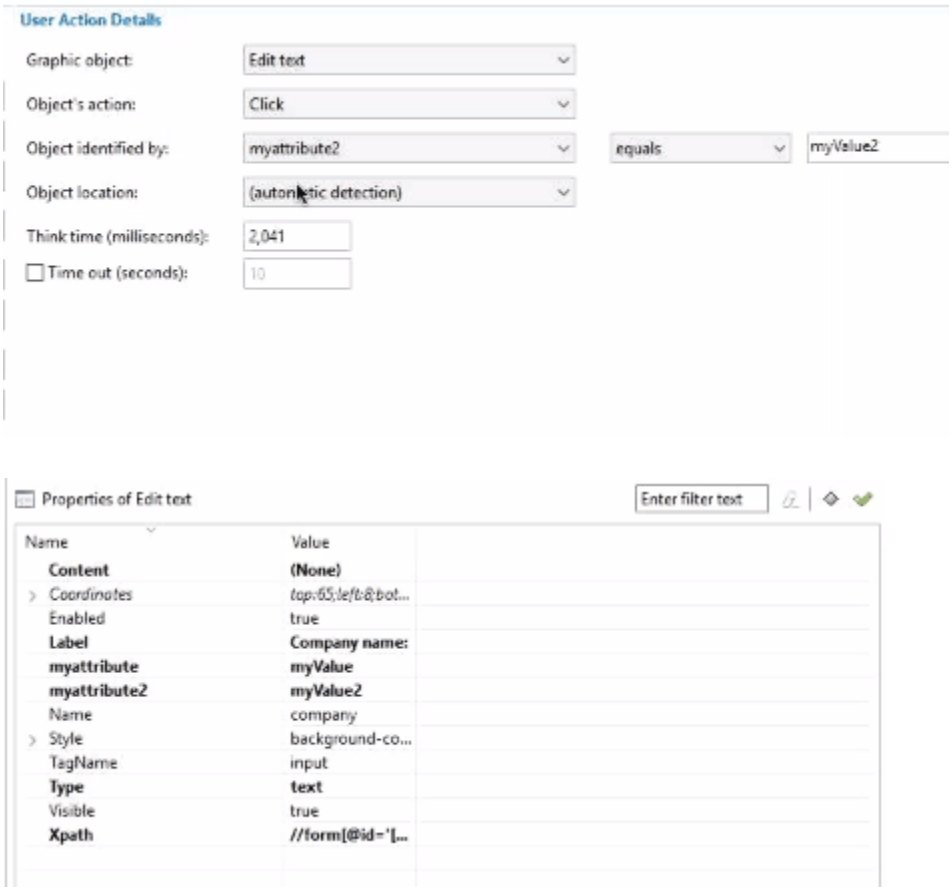
By default, the Web UI tests use the standard object properties such as Content, ID, and Class, Xpath to find, identify and locate objects for UI controls. But some applications under test use custom attributes to describe properties for UI controls. To avoid standard behavior, you need to add them to a list of preferred identifiers and prioritize them so that they are recognized as the main object identifiers in the application under test.

In the preferences, set a list of these properties:

1. In the test editor, click **Window > Preferences > Test > Recording > UI Test Recording > Preferred object identifiers**
2. Click **Add** and in the dialog that opens, enter a name for the web object property so that it can be used as an object identifier when the test is recorded and run. Then, click **OK**
3. Use **Up** and **Down** buttons to prioritize the attributes in the list. Click **Apply** and **OK**



4. When the list of preferred object identifiers is set, record the test. If the test was recorded before you set a list of preferred object identifiers, you need to run it again to apply the preferences.
5. Open the test script, and check that the attributes are available in the list of properties in **Object identified by**.



Results

Now, the custom attributes are automatically recognized when recording the test, they are listed as main object identifiers in **Object identified by** and in bold, in the table of properties. You can modify the property and its value on the test steps on page 601 as for the standard attributes.

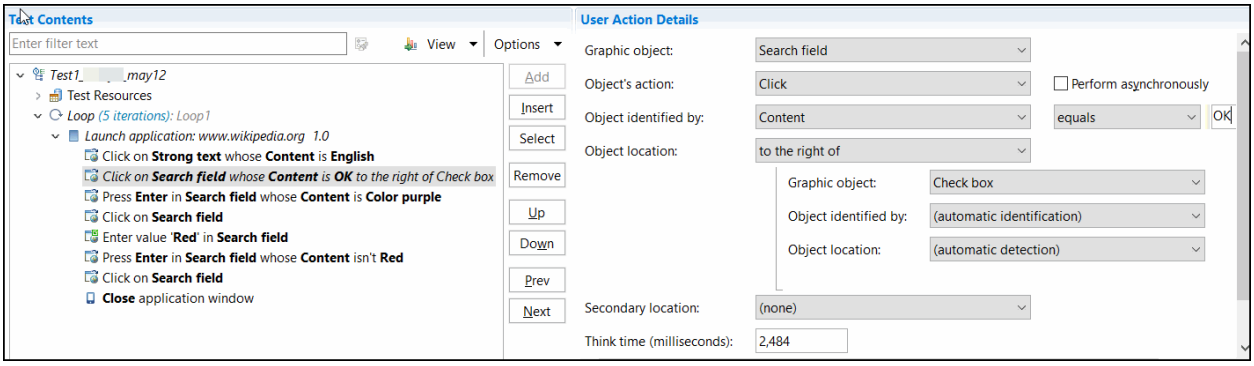
Object location in a test

During a Web UI test playback, automatic detection of graphical objects in an application under test (AUT) is essential. However, IBM® Rational® Functional TesterHCL OneTest™ UI might not be able to locate a specific object on which an action needs to be performed for various reasons. In such cases, you can update the test script and provide more accurate information to locate the target object.

Rational® Functional TesterHCL OneTest™ UI provides a variety of approaches to identify and locate objects effectively within the AUT. For example, if a recorded step instructs the test to *click on Edit text whose content is 'August 30th, 2013'*, the test might fail during the playback if the current date is different. To address this issue, you can modify the test script to provide more precise object location details. By doing so, Rational® Functional TesterHCL OneTest™ UI ensures that the object is located correctly during the test playback, regardless of dynamic factors.

In Rational® Functional TesterHCL OneTest™ UI, various object location operators are available to identify objects in an AUT. They are displayed in the **Object location** fields in the **User Action Details** panel of the test editor. See Setting object location conditions in a test script on page 606.

The following screen capture displays fields related to object location in the test editor:



Object location in Rational® Functional TesterHCL OneTest™ UI is a critical aspect to enhance object identification during the test playback. By updating test scripts with precise object location details, you can overcome challenges that are associated with dynamic content or changing application states.

Setting object location conditions in a test script

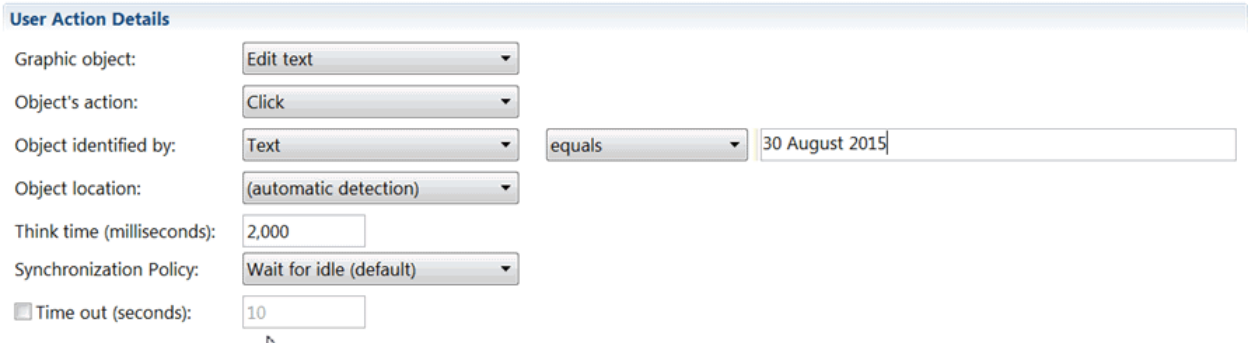
In some cases, recorded actions are not replayed as expected because the objects cannot be found. In the test, multiple location operators are available to improve object recognition.

About this task

For additional details about object recognition in mobile and web UI tests, see Enhancement of test script robustness on page 599.

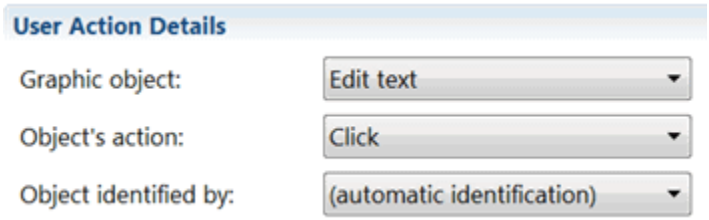
Here is an example where setting object location conditions is helpful: You record a test, and the action on a step, Click on Edit text,, is not clearly identified, with no label, and with variable content, such as a date. When the test is played back, the action cannot be performed because the date has changed.

Figure 3. Example of action that can fail when the test is played back.



For this test to play back successfully, you can modify the object location in the test script so the target object can be found during test replay. For example, you can indicate that **Edit text** is to the right of a stable graphic object that is easily identifiable, for example, an **Edit text** field whose label is **city**. You can proceed as follows:

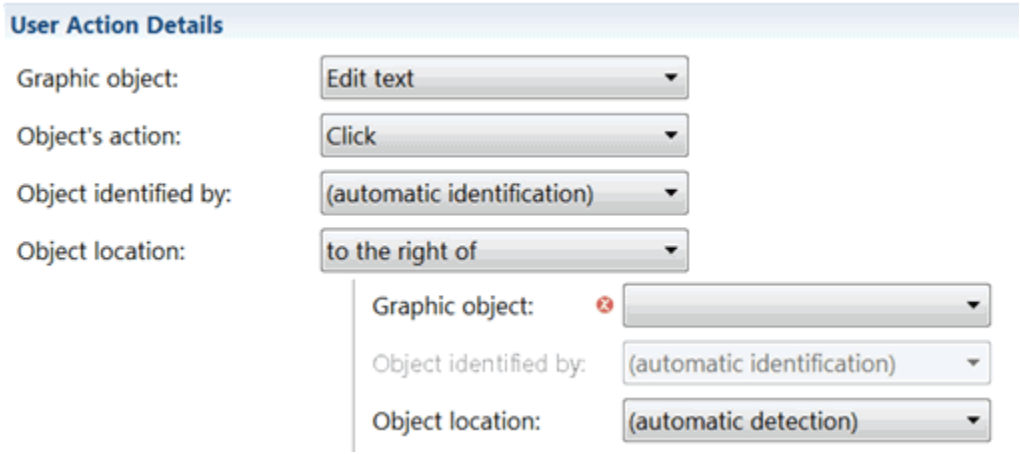
1. In the test, click a test step.
2. In the **User Action Details** area, set **Object Identified by** to **(automatic identification)**.
   Figure 4. Automatic detection



3. Select a location object in the **Object location** field.

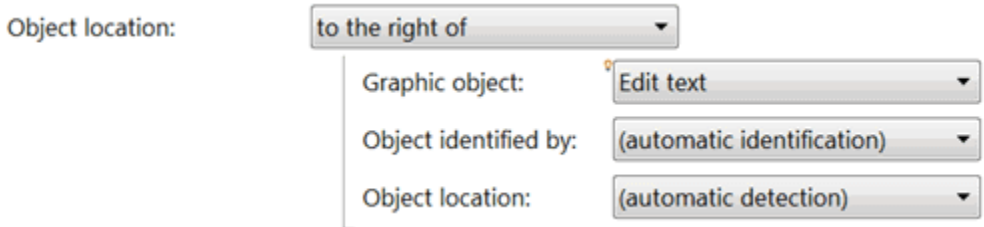
   The **Object location** area expands to include indented fields for **Graphic object**, **Object Identified by**, and **Object location**.

   Figure 5. Select an Object location



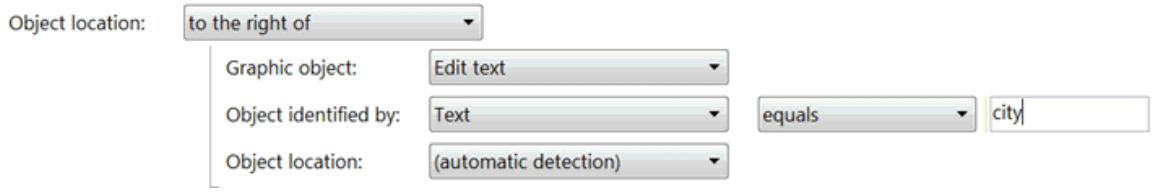
4. Select the graphic object that will be used as the reference object in the indented **Graphic object** field, for example, **Edit text**.

   Figure 6. Select a reference object



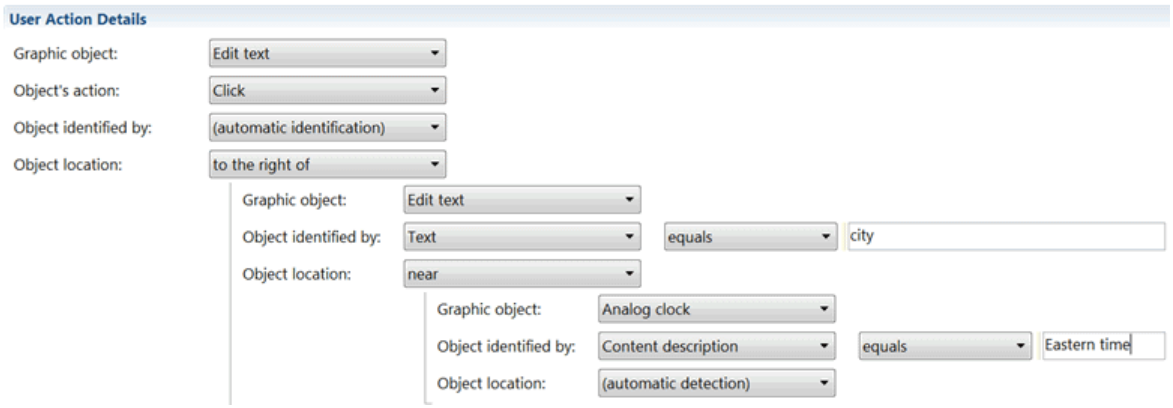
5. Select its property, for example, **Text**, in the indented **Object Identified by** field and enter its value, for example, `city`.

Figure 7. Select a property and enter a value



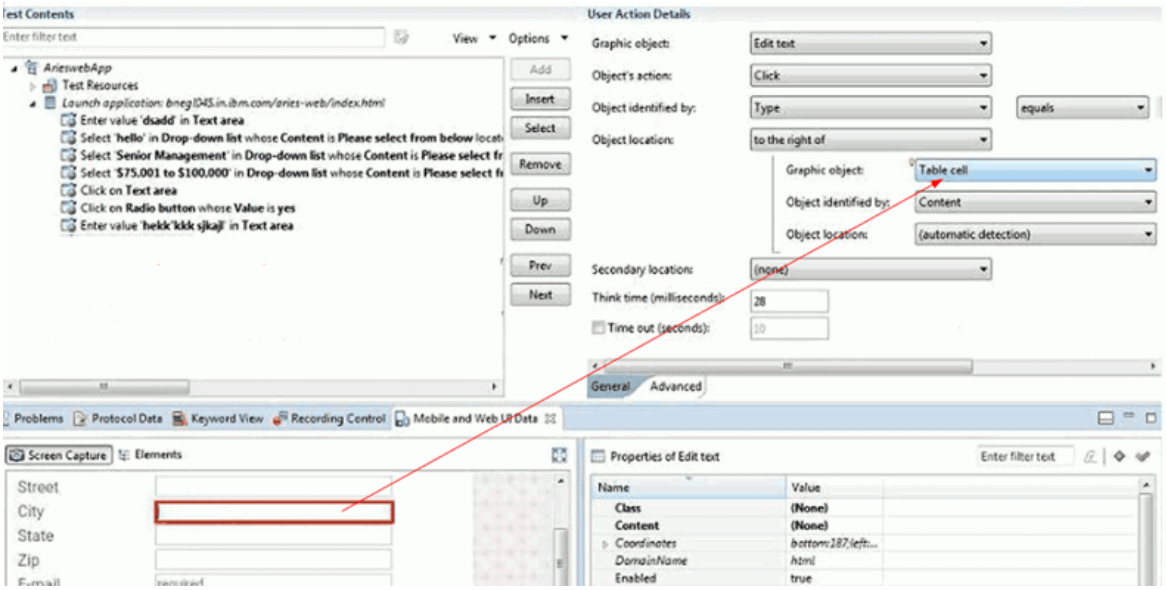
You can verify the property description of the object in the Properties table or in the **Elements** tab of the **UI Test Data** view.

6. Define another object location that is helpful to find the reference object. You can indicate, for example, that the object whose value is `city` is located near an **Analog clock** whose content is `Eastern time`.
   Figure 8. Set an object location condition with a location operator, graphic object, its properties selected and values



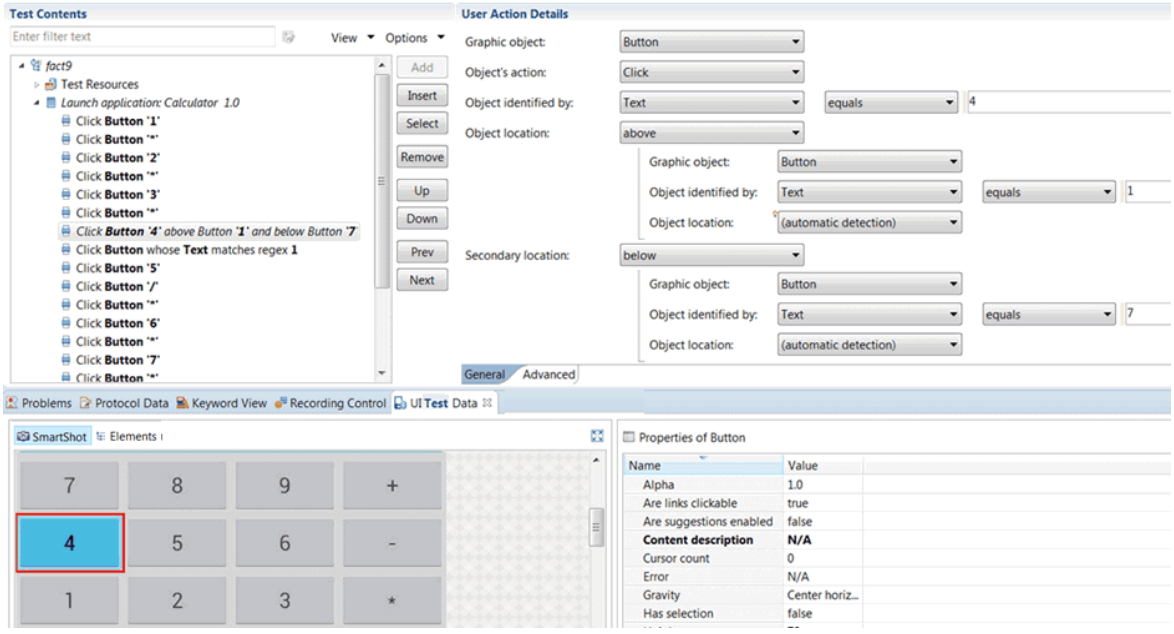
7. Save and run the test to verify that the step is successful.
8. Another method of setting object location conditions is to select an object in the **Screen capture** tab, and to drag it on the **Object location** field. That way, the property and value of the object selected are automatically entered in the object location fields.

Figure 9. Drag the object and drop it on the Graphic Object field



9. You can also set a secondary locator condition to identify the target object on the step. The object can be found easily if the conditions are met. This could be useful, for example, in a test using a spreadsheet or a calculator. To set a secondary condition, you proceed as explained above. In the Secondary location field, select a location operator, a graphic object, the object property in **Identified by**, an operator, and enter the required values.

   Figure 10. Set a secondary location condition, with a location operator, a graphic object selected, its properties and values.



10. Save and run the test to verify that the step is successful.

Results

When all of the steps play back successfully, the test can be used in an automated testing process.

Related information

Modifying the property that is used to identify the object in a test on page 601

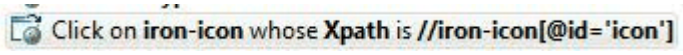
Image recognition in a test

In Web UI tests, IBM® Rational® Functional TesterHCL OneTest™ UI must identify objects accurately for a successful test playback. While most objects can be identified using text properties, there are cases where text-based identification might not be feasible or effective. To address this challenge, Rational® Functional TesterHCL OneTest™ UI provides image recognition capabilities so that you can identify objects based on their visual appearance.

To address any potential challenges with image recognition, Rational® Functional TesterHCL OneTest™ UI uses image correlation techniques to recognize and manage objects during playback. During test recording, Rational® Functional TesterHCL OneTest™ UI captures an image (reference image) that represents the action that is performed. This reference image is then compared with the actual image (candidate image) of the AUT during playback. A recognition threshold is used to accept an adjustable rate of differences between the reference image and the candidate image, and to evaluate if the images match. The default recognition threshold is set to 80, and the default tolerance ratio is set to 20.

Image correlation can be used in various test scenarios. Following are some examples:

- Suppose you record a test on a mobile phone and play it back on a desktop. Because the image width and height differ from one device to another, the test playback might fail on devices that do not have the same screen ratio. You can use image correlation to address this issue.
- In certain cases, the target objects within a test recording might undergo changes when the test is played back. For instance, when a virtual keyboard is used in a secure application, the position of digit buttons can vary from one server session to another. Using image correlation, Rational® Functional TesterHCL OneTest™ UI can adapt to these dynamic changes.

Note: From Rational® Functional TesterHCL OneTest™ UI v9.1.1, custom graphic objects are recognized. In the edited test, a custom object is identified as a **Custom Element** graphic object, with the **name1-name2** description in the test script 

.

During a test playback, if Rational® Functional TesterHCL OneTest™ UI fails to locate a certain image within the AUT, or if you have selected an inappropriate image for image identification, the test can fail. To address recognition issues during playback, you can modify the image that is used for target object identification in the test step. Also, you can adjust the threshold score and tolerance ratio in the edited test to enhance image recognition. If you set the threshold to 0, the candidate image that is most similar to the reference image is selected, even if they are not identical. Conversely, if you set the threshold to 100, even a slight difference in images might prevent image recognition.

When you set the recognition threshold in the test editor, Rational® Functional TesterHCL OneTest™ UI displays an *image matching preview* view to help you find accurate candidate images for test playback. The best matching candidate images are highlighted in green. Images that have a score that is above the threshold are highlighted in yellow and might not be the most suitable. And candidate images that have a score that is below the threshold are highlighted in red, which indicates that they do not match the reference images. You can find the image correlation details in the test report that is displayed when a test execution is complete.

You can add images as the main property in a verification point on the target objects in the tests. For example, you can verify the position of a dropdown list on a screen. For details, see Creating verification points from the SmartShot View on page 584.

By using the image recognition feature, Rational® Functional TesterHCL OneTest™ UI can identify objects based on their visual appearance when the text-based identification is a challenge.

Responsive design condition

Applications that use the Responsive Design feature introduce variability into their screens. This variability can result in test failures due to factors such as changing screen layouts, the presence of user-specific interactions such as location prompts or tutorials, and so on.

To overcome test failures that are caused by Responsive Design elements, you can define execution conditions for a selected set of actions in IBM® Rational® Functional TesterHCL OneTest™ UI. This allows you to specify blocks of actions that are executed only during the initial run of the test. The block of actions is ignored on subsequent test runs, therefore preventing step failures.

Note: The Responsive Design condition feature is currently available for Android applications in Rational® Functional TesterHCL OneTest™ UI version 8.7.1 only.

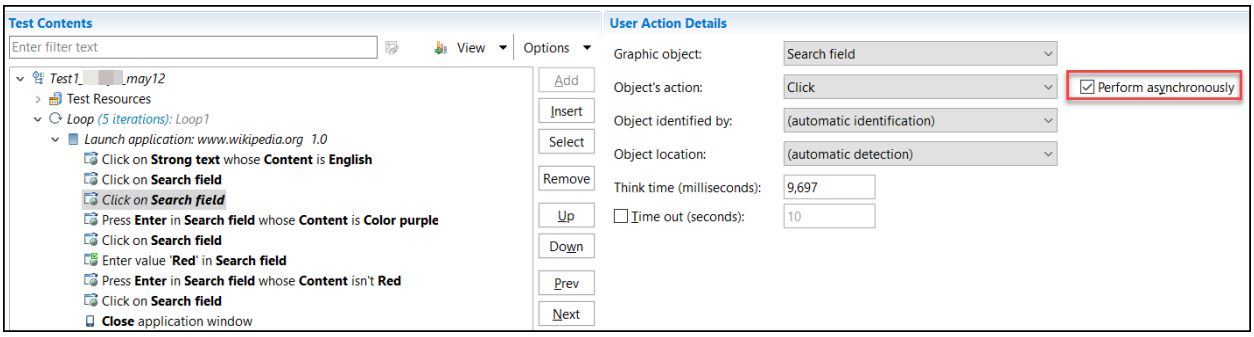
To summarize, the Responsive Design feature is found in many applications where the behavior and visual elements adapt to the device that is used. You can create Responsive design conditions in an Rational® Functional TesterHCL OneTest™ UI test to avoid test failures. See Creating Responsive Design conditions in a test.

Actions performed asynchronously

During a Web UI test playback, certain actions such as *click*, *hover*, *press Enter* might synchronization issues and result in step failures. To address this challenge, you can use the IBM® Rational® Functional TesterHCL OneTest™ UI feature to perform certain actions asynchronously.

In a recorded Web UI test, when you select a step in the **UI Test** view, the **User Action Details** panel displays the object properties that are associated with the action performed in the step. In the **User Action Details** panel, you can select the **Perform asynchronously** option beside the **Object's action** field so that the selected action is performed asynchronously during the test playback.

Note: This feature is available from Rational® Functional TesterHCL OneTest™ UI version 9.2.1 onwards.



By enabling the asynchronous execution for actions in Web UI tests, you can minimize the synchronization issues during the test playback.

Identification of dynamic window titles

During a Web UI test playback, it is difficult to identify dynamic titles in multiple windows. The titles can change dynamically based on factors such as date, time, or other variables. To address these dynamic changes, you can use the regular expressions in IBM® Rational® Functional TesterHCL OneTest™ UI.

During a Web UI test recording, there can be scenarios where a report or information is displayed in a new window. The title of this window can contain dynamic information such as the current date and time. In such cases, you can edit the recorded test and use regular expressions in the title field of the **Window** node that matches the pattern of the dynamic title of the window.

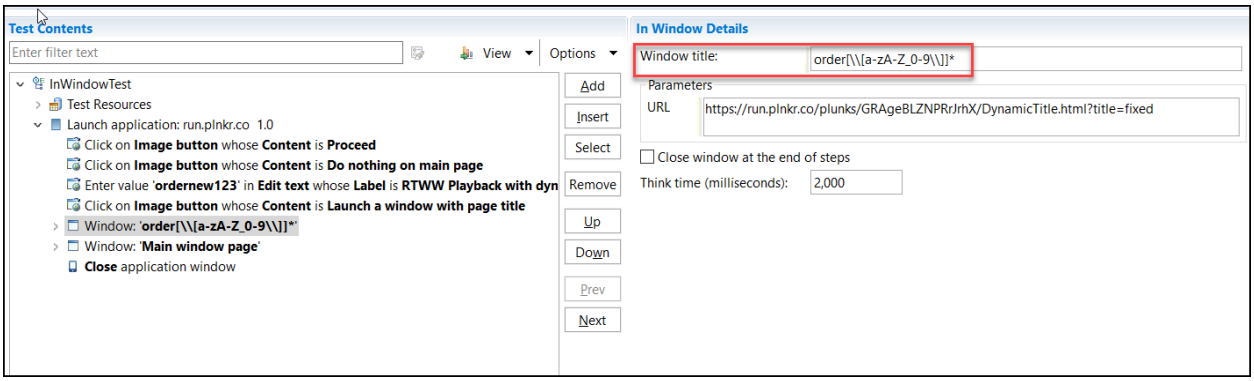
Consider the following example.

During a Web UI test recording of an application under test (AUT), a report was displayed in a new window, and the new window (report) displayed the following title.

Window title: Order_2023May19T143000_[AzHrlo]

In this scenario, the report title contains dynamic data that includes alphanumeric characters, underscores, and brackets, which changes every time the report is generated when you play back the test. For Rational® Functional TesterHCL OneTest™ UI to identify this dynamic title during the playback, you can use the following regular expressions in the **In Window Details** panel of the corresponding **Window** node in the test.

Regular expressions used in the Window title: Order[\\[a-zA-Z_0-9\\]]*



By using regular expressions for the title of dynamic windows, Rational® Functional TesterHCL OneTest™ UI can identify and locate the window during the test playback. The regular expressions act as dynamic patterns that match the changing elements within the title. Therefore, Rational® Functional TesterHCL OneTest™ UI can accurately identify multiple windows, even when there are changes in the title.

Substituting the URL of an application by using Datasets

If you have the same Web UI application deployed on different servers corresponding to different phases of a project (development, production, etc.), or different versions of the application for example, you might want to execute the same Web UI test on all these servers at once. To switch from one server to another, you can edit the URL of the application and run the test on a different server. To execute tests on different environments at once, you can substitute the URL from a dataset so that you can run the same test in a loop on different servers.

About this task

This task applies to Web UI applications only, from the UI Test perspective.

1. To switch from one server to another one while you are running your test, you need to edit the application URL as follows:
    a. In Rational® Functional TesterHCL OneTest™ UI, open the test in the **Test Editor**, and then click the **Launch application** root step.
    b. In the **Application Details** pane, select the address in the **URL** field, and then enter a new URL, or modify the current URL. Example: Replace the . *com* extension with the *.en* extension.
2. To execute a test in different environments, use a dataset to substitute the value in the **URL** field, with a variable test data, for example, you can replace the .com extension of the URL with the .en extension, as follows:
    a. Create the dataset. For more information about this subject, see Creating a dataset associated with a test on page 637.
    b. Associate the address field request in the test with a particular column in the dataset. To do so, select the URL, right-click the URL in the field, and select **Substitute > Select Data Source**. For more information about this subject, see Associating a dataset with the test.

c. Once the dataset and column are selected, click **Finish**. In our example, there is a compound test with two iterations of the test, one for the .en site, the other for the .com site.
d. Run the test. During the playback, you will have to check that the first iteration is replaced with the second one.

Note: In a similar way, you can click the **In Window** step of the test, and then substitute the entire URL or part of the URL in the **In Window Details** pane.

Extension of application URL in Web UI tests

You are familiar with the feature to substitute text in the application URLs of Web UI tests. Starting from 10.1, you can extend the application URLs with substituted text from one Web UI test to other Web UI tests. This feature reduces your effort of manually substituting the application URLs in all your tests that have an application URL.

After you substitute a partial or an entire application URL used in a Web UI test for a particular phase of your project, you can extend the substituted application URL to the tests used in other phases of the project.

For example, if you have a project called *"abc"*, then the different phases of the project might have the following application URLs:

- https://www.abcdev.com
- https://www.abcprod.com
- https://www.abcnonprod.com

You can also have different tests in each phase that tests different pages with the same primary URL. For example, https://www.abcprod.com/welcome.htm, https://www.abcprod.com/intro.htm.

If you want to change any part of the application URL, then you have to manually substitute the application URL in each test that uses the URL. This feature enables you to substitute the text in the application URL in any one test. You can then extend the substitution in the application URL across all tests that use the URL.

You can use different data sources that contain text values to replace the text in the URL. You can use the following types of data sources:

- Built-in variable
- Custom Java code
- Dataset
- Test variable

You can extend the substituted URL from a test to other tests by using either of the following methods:

- Extending the application URL from the Application Details pane on page 615
- Extending the application URL from the UI Test applications view on page 616

Restriction: The extension of application URL in Web UI tests does not work as usual when tests match the following conditions:

- The tests have a variable that is shared
- The values of the shared variable are different in each test

The value of the shared variables in the application URL that you extend does not get updated in the application URL of the tests that receive the extended application URL.

For example, you have declared a variable *VAR* that you share between two tests, A and B. The value of the variable *VAR* is set as `value1` for test A, and `value2` for test B. If you substitute the variable *VAR* in the application URL in test A, and then extend the application URL to test B, the value of the substituted variable in the application URL of test B remains at `value2` and is not replaced.

Extending the application URL from the Application Details pane

After you substitute the text in the application URL of a Web UI test, you can extend the substituted application URL to other Web UI tests from the **Applications Details** pane.

1. Select the UI Test perspective from Rational® Functional TesterHCL OneTest™ UI.
2. Select the Web UI test from the **Test Navigator** pane for which you want to substitute partial or entire application URL.
3. Select **Launch application** from the **Test Contents** pane.

   The **Applications Details** pane is displayed.
4. Go to the **URL** field, select and right-click the text in the URL that you want to change.
5. Click **Substitute > Select Data Source**.

   The **Select Data Source** dialog box is displayed.
6. Select the data source and click **Select**.

   The URL text is substituted with the text in the data source and is highlighted in the **Application Details** pane.

   Note: You can substitute different parts in the application URL with text values in different data sources.

7. Click the **Replace test application using substituted application** icon.

   The **Replace test application using substituted application** dialog box is displayed that contains the Web UI tests with an application URL.

8. Select the tests for which you want to replace the application URL with the URL that contains the substituted text.
9. Click **Finish**.

Results

You have successfully extended the application URL with the substituted text to other Web UI tests from the **Application Details** pane.

Extending the application URL from the UI Test applications view

After you substitute the text in the application URL of a Web UI test, you can extend the substituted URL other Web UI tests from the **UI Test applications** view.

1. Select the UI Test perspective from Rational® Functional TesterHCL OneTest™ UI.
2. Select the Web UI test from the **Test Navigator** pane for which you want to substitute partial or entire application URL.
3. Select **Launch application** from the **Test Contents** pane.

   The **Applications Details** pane is displayed.
4. Go to the **URL** field, select and right-click the text in the URL that you want to change.
5. Click **Substitute > Select Data Source**.

   The **Select Data Source** dialog box is displayed.
6. Select the data source and click **Select**.

   The URL text is substituted with the text in the data source and is highlighted in the **Application Details** pane.

   Note: You can substitute different parts in the application URL with text values in different data sources.

7. Click the **Update Web UI Application** icon in the **Application Details** pane.

   The **UI Test applications** view is displayed.

   You can view the following changes:
   - The application URL that contains the substitutions is selected by default.
   - The list of tests available to the application URL is displayed in the **Available Tests** pane.
   - The tests that contain a different URL or substitution when compared to the URL that you want to extend are displayed in italics.
8. Select the required tests in the **Available Tests** pane for which you want to replace the application URL with the URL containing substitutions.
9. Click the **Update application in selected test suites** icon.

The **Replace application in test suites** dialog box is displayed.

10. Click **Finish**.

Results

You have successfully extended the application URL with substituted text to other Web UI tests from the **UI Test applications** view.

Overview of guided healing and self-healing

As web applications undergo changes during regression testing, the application GUI is frequently modified and the regression tests must be updated to identify possible issues, and to ensure that the new code has not affected other parts of the software. Some changes can be easily detected such as edit box labels, but others can be more difficult to identify, for example, the UI control properties. To help testers to identify those changes in the application and to fix the Web UI tests accordingly, more intelligence has been added to test playback with the guided healing and self-healing features.

You can apply guided healing and self-healing features when you initiate tests through the following interfaces:

- UI Test perspective of Rational® Functional TesterHCL OneTest™ UI
- Command line of Rational® Functional TesterHCL OneTest™ UI

Guided healing and self-healing through the UI Test perspective

To enable the guided healing feature, you must select the **Collect data to update test steps** option when you run your test. Thus, data is collected during the test run. If an object is not found, Rational® Functional TesterHCL OneTest™ UI identifies the object that best matches its description as a possible candidate to replace the missing element and a snapshot is taken. When the test is complete, the steps are highlighted with colors to identify which of the test steps passed and those that require updates. Then you can check the status of each step in the edited test and update the steps manually.

Another option is available from Rational® Functional TesterHCL OneTest™ UI v10.0.0 to enable automatic update of the tests. It is the **Automatically update the test after the end of the execution** option in the **Run configuration** wizard that is known as the self-healing option. It must be selected when you run your test and enable the guided healing option. Thus, when the test execution is complete, the test steps are automatically updated and the tester has no change to make. It applies to Web UI tests and also to compound tests from Rational® Functional TesterHCL OneTest™ UI v10.0.1. When you choose this option for a compound test, all the Web UI tests are self-healed.
The hierarchy and snapshots are captured. But at the end of the execution, the amber steps are automatically updated. The self-healing feature is particularly useful when you have numerous test cases that remain re-usable for successive regression cycles.

Updating tests manually can be carried out as a first testing phase prior to starting using the self-healing feature. You can update tests manually first to verify that the test steps are updated with the best candidates identified during the playback. You can then proceed with updating tests using the automation feature.

Self-healing through the command line

You can run tests with the self-healing feature enabled from the command line by using the **protocolinput** option with the *autoheal* attribute set to true. The autoheal attribute supports collection and auto-modification of the test data.

The sample syntax is as follows:

cmdline -workspace <workspacename> -project <projectname> -suite <suitename> -exportlog <exportlogpath> -results autoresults -protocolinput "autoheal=true"

Updating tests by using the guided healing feature

You can update the UI changes in a Web UI test by using the guided healing feature that collects the required data during a test run. When the test run is complete, you can view the status of each step in the edited test. Also, you can verify whether the collected data is appropriate or not to replace the objects that need to be updated, and then make replacements.

Before you begin

You must have completed the following tasks:

- Read about the guided healing feature. See Overview of guided healing and self-healing on page 617.
- Recorded a Web UI test. See Recording a Web UI test on page 545.

About this task

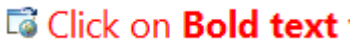
You must perform this task from the UI Test perspective. This task applies to Web UI tests.

Rational® Functional TesterHCL OneTest™ UI collects Web UI data during the test run and snapshots taken during the test run are used to update the failed steps. If an object is not identified during the test run, the object whose description matches the best is selected as the possible candidate. The test steps and the images highlighted in the test steps are indicated in different colors to specify the status of each test step.

Note: You must set enough timeout for each test step before running the test to get proper snapshots and enable the guided healing option. If a test step fails with Time Out Error, because of low time out values, the snapshot of the web page might not be captured.

You can set the timeout for each step by selecting the **Time out** checkbox and entering the value in seconds in the **User Action Details** view.

You can refer to the following table to understand the meaning of step results displayed in different colors:

The default color code of step results	Description	Indication of the test result
Green For example, the description of the test step is displayed as follows: 	During the playback, the target object that matched the actual object was found based on the correlation score.	The test step is passed.
Amber For example, the description of the test step is displayed as follows: 	During the playback, the target object that matched with the actual object was not found. Therefore, the next closest image that matched was identified as the target object based on the correlation score.	The test step is partially passed and you must update the step.
Red For example, the description of the test step is displayed as follows: 	During the playback, no target object matched the actual object based on the correlation score.	The test step is failed.

Note: You can modify the default colors of test step results in **Window > Preferences > Test > UI Test > Test Step Results**.

1. Click **Run Test** to run the recorded Web UI test from the test editor.
   The **Run Configuration** dialog is displayed.
2. Select the **Collect data to update test steps** checkbox if it is cleared.
3. Click **Next**.
   The **Advanced playback options** page is displayed.
4. Select the options as required for the test run, and then click **Next**.
   The **Performance Measurement** page is displayed. The options on this page are applicable only for mobile tests and they are disabled for Web UI tests.
5. Click **Finish**.
   Result

The test run begins with the guided healing feature enabled. After the test run is complete, the Web UI test report is displayed in the browser.

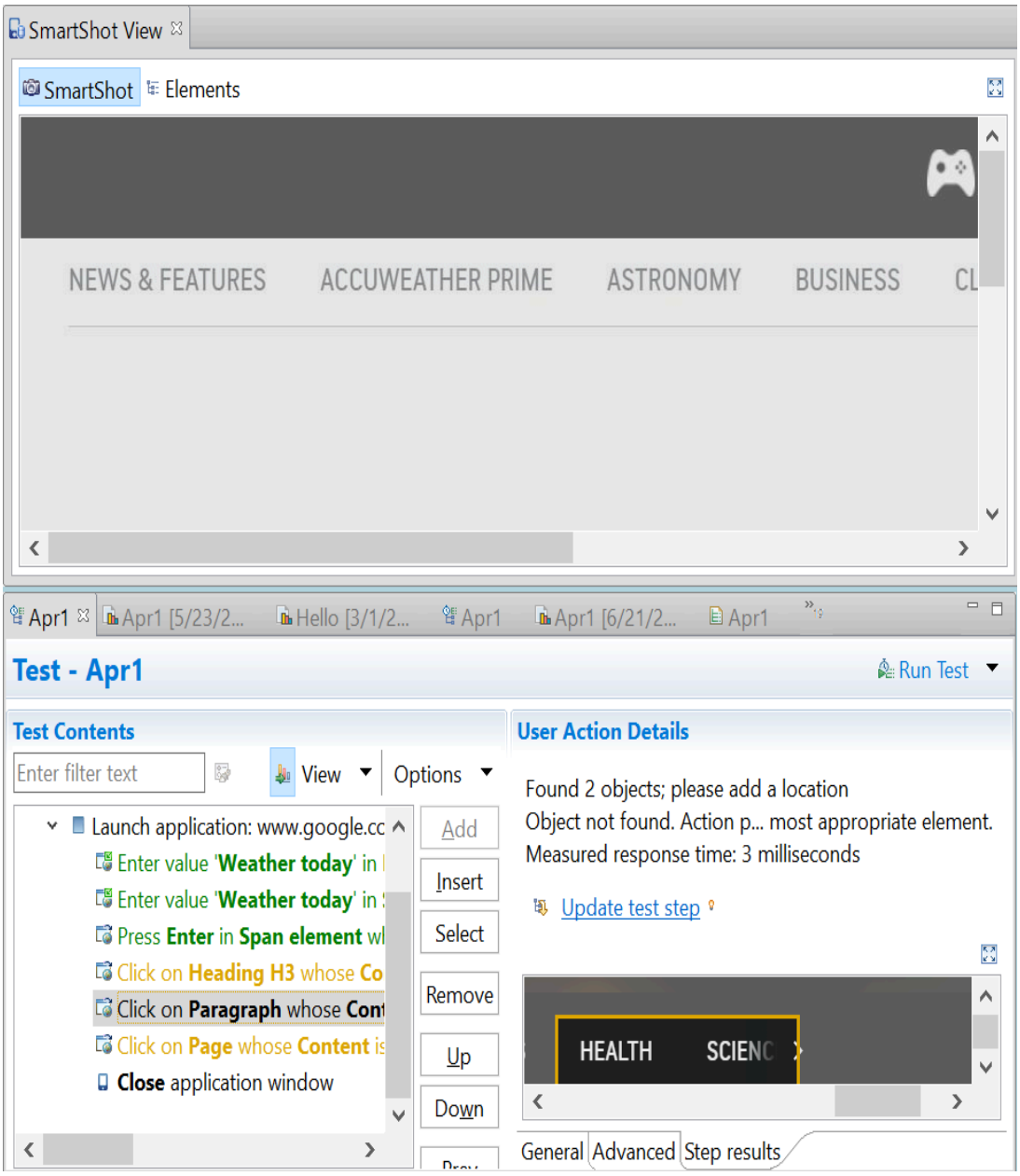
You can click a test step in the **Details** panel to verify whether the step is passed through the guided healing option or not. The test steps for which the guided healing is applied are indicated with the following message in the report:

```
Object not found. Action performed on the most appropriate element.
```

6. Perform the following actions in Rational® Functional TesterHCL OneTest™ UI to update the test steps for which the guided healing is applied:

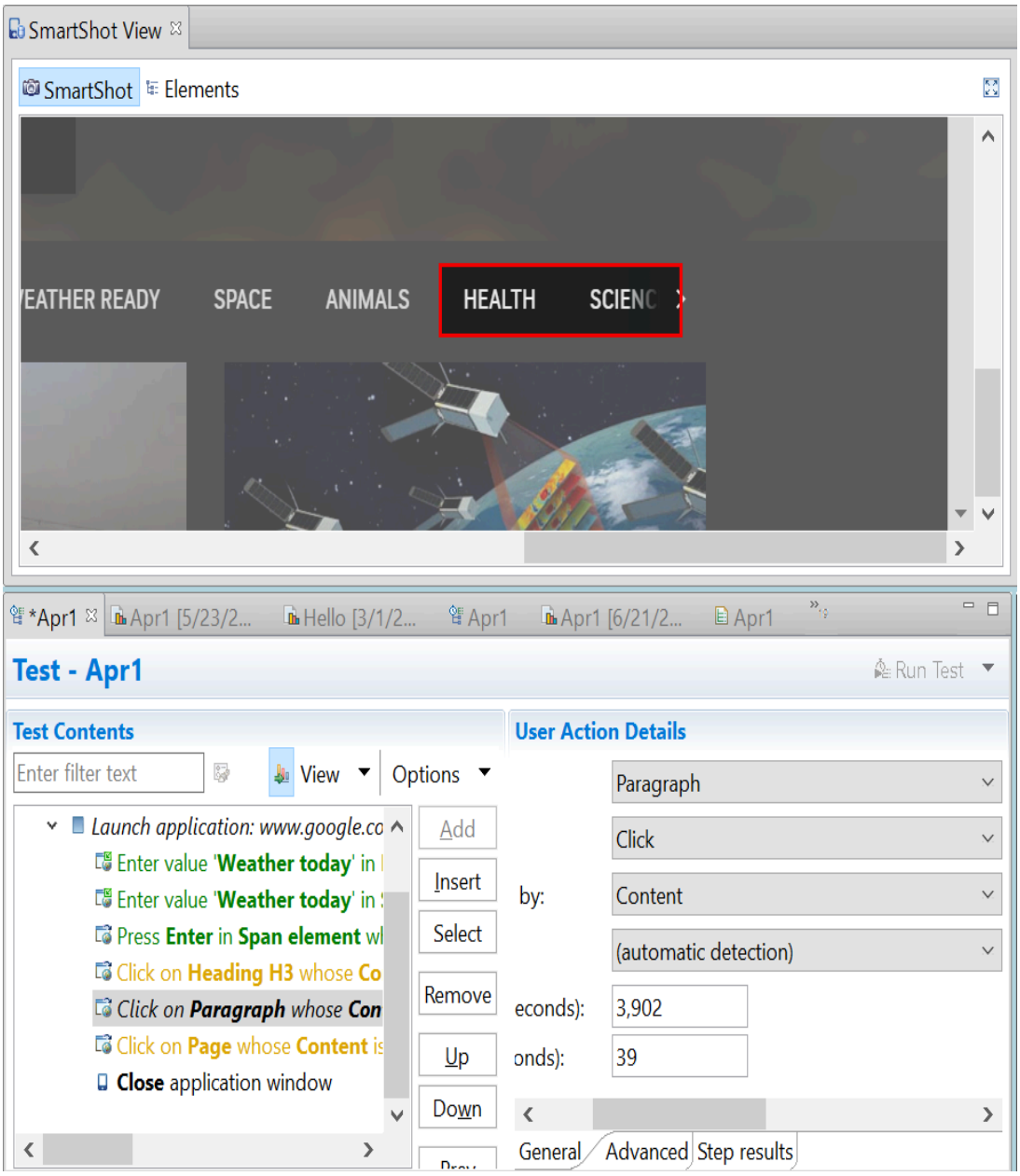
    a. Click the **Scan and mark test with latest result**  icon in the test editor to view the status of the test steps.
    b. Click the step that is in amber color.

    The snapshot that is captured during the playback is displayed in the **User Action Details** view under the **Step results** tab.

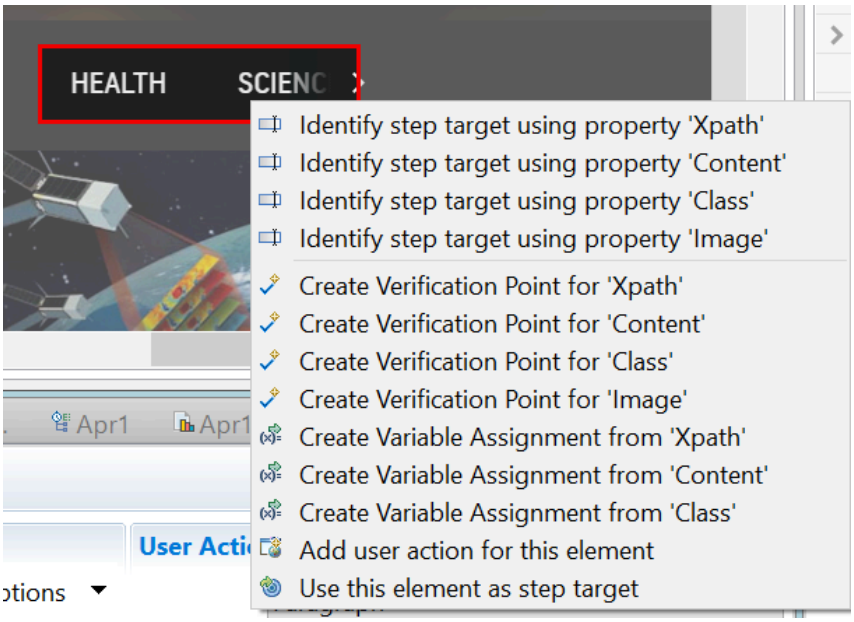


c. Click **Update test step** to update the test.

The new captured snapshot is copied to the **SmartShot View** and is displayed under the **SmartShot** section. The snapshot includes the control that has a user action for the step.



d. Right-click the control and select **Use this element as step target** to update the test step as shown in the following example:



e. Repeat the steps from 6.a on page 620 to 6.d on page 622 for any other test steps in amber color that were caused by changes in the UI of the application under test.

7. Save the test.

Results

You have updated the test by using the guided healing feature and the updated test can be run with no failures.

What to do next

When the snapshots are retained, the test file size increases. Therefore, you can remove the snapshots when the test development is complete. To remove the snapshots, you must right-click the test in the **Test Contents** window, and then click **Delete snapshots and hierarchies**.

Note: If you want to retrieve the snapshots taken during the last test run, you must select the step for which you need to retrieve the snapshot, right-click and select **Update test steps**. The change applies only to the selected step.

Related information

Editing Web UI tests on page 573

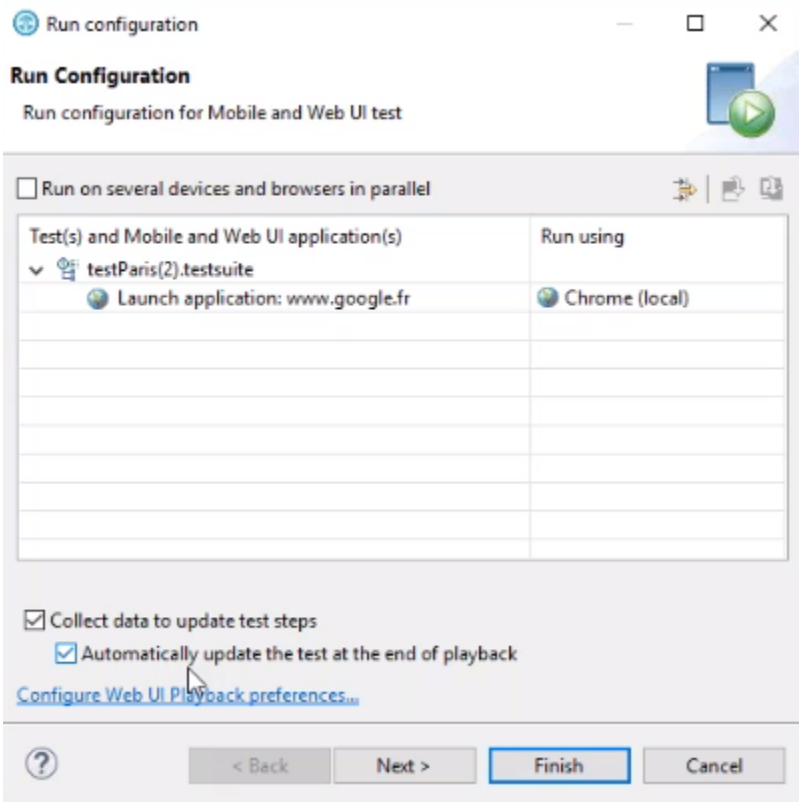
Updating tests with the self-healing feature

From Rational® Functional TesterHCL OneTest™ UI 10.0.0, your Web UI tests can be self-healed, and it also applies to compound tests from 10.0.1. You just have to select an option when you run your test and no more action is needed. Data is collected during the test run and test steps are automatically updated at the end of the test execution.

About this task

This task must be performed from the UI Test perspective. The self-healing feature applies to Web UI tests and compound tests.

1. Open your test in the **Test editor**, and click **Run**.
2. To enable guided healing feature, select **Collect data to update test steps** in the **Run configuration** dialog box.
3. To enable automatic updating of the test, click **Automatically update the test after the end of the execution** and click **Finish**.



Results

The test is updated and can be run with no failures.

Validating images and user interface elements by using the image property

When you play back the Web UI test, you can validate the images and user interface elements by the using the image property. After you define a test step to use image property, during playback, Rational® Functional TesterHCL OneTest™ UI locates and compares the selected image with the target objects in the web pages and finds the best matching image.

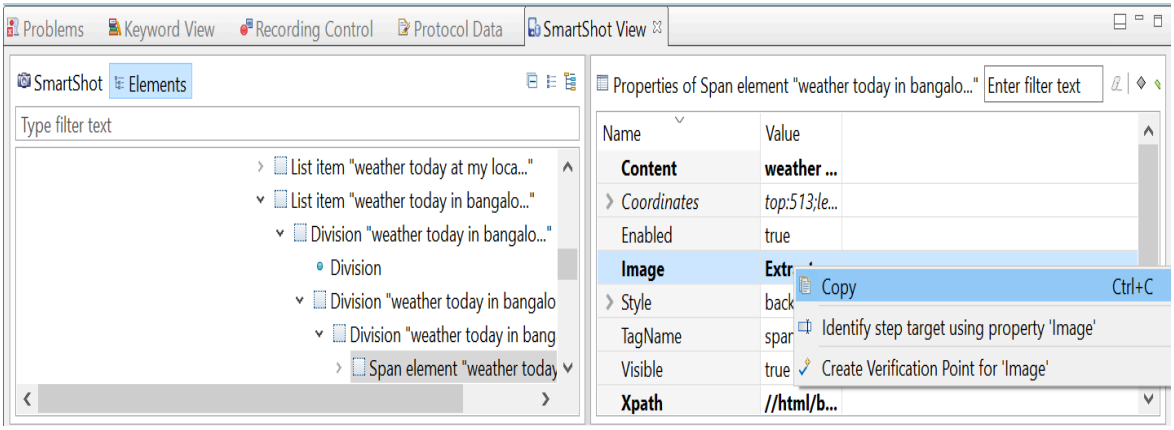
Before you begin

You must have recorded Web UI tests that contain images or user interface elements such as input controls, navigational components, and so on.

About this task

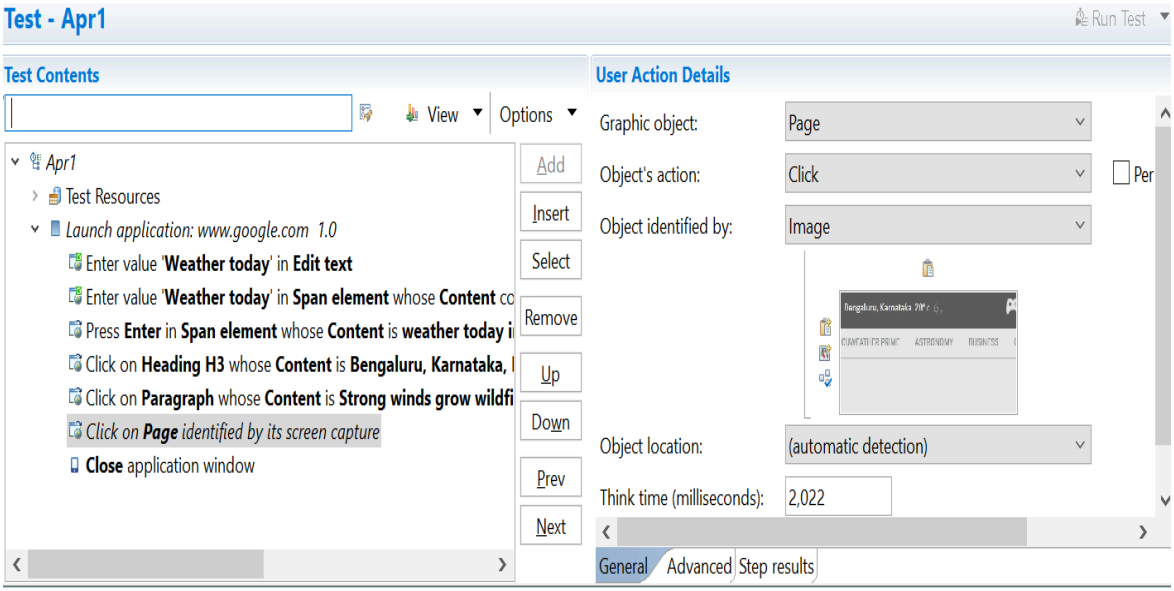
While recording a Web UI test, the screen capture of the images and user interface elements are captured in their original appearance. The images and web controls are captured before there is any change in their appearance caused because of cursor movements or keyboard inputs. When you play back the Web UI test, these images in their original appearance are used to identify the target objects appropriately. The original appearance of the images in turn improves the correlation score that is displayed in the report.

1. Open the recorded Web UI test.
2. In the **Test Contents** pane, click a test step that contains images or user interface elements. The related properties of the captured object are displayed in the **Properties** pane.
3. Right-click the Image property in the **Properties of <object>** pane in **Smart Shot View** to select the following options:



Option	Description
Copy	Copies the image that is selected in the **ScreenShot** pane to the clipboard.
Identify step target using property 'Image'	Updates the test steps to recognize the target object during the playback by using the selected images. **Note:** The selected image can be changed by using the **User Action Details** pane.
Create Verification Point for 'Image'	Creates a verification point for the test step by using the image captured in the image property.

4. Optionally, perform the following actions to change the image property by using the image editor in the **General** tab of the **User Action Details** pane:



To do ...	Click ...
Update an existing image with an image from the clipboard to compare it with the target objects and find the best matching image.	
Add multiple images in addition to the image that is already available in the image editor. When the appearance of the target object varies based on the platform and browser, you can use this option to recognize your target object. During the playback, Rational® Functional TesterHCL OneTest™ UI searches for the best matching image based on the correlation score. This score is displayed in the UI Test report	
Add a new image from the local drive of your computer to compare it with the target objects to find the best matching image.	
Set the recognition threshold and the aspect ratio tolerance parameters that define the match criteria. Recognition threshold value specifies the match percentage of the images. For example, if the Recognition threshold is 80%, then the target objects that match the source image up to 80% of pixels are considered as a match.	

To do ...	Click ...
Aspect ratio tolerance value specifies the allowed deviation percentage of the height-width ratio of the images. For example, if the aspect ratio tolerance is 20% and the dimension ratio of the selected image is 1, then the target objects whose ratio varies from 0.8 to 1.2 are considered as match. **Note:** By default, the Recognition threshold is 80% and the Aspect ratio tolerance is 20%.	

5. Save the changes.
6. Run the Web UI changes.

   Result

   You have successfully configured the image and user interface elements in a recorded Web UI test. The image and user interface elements are validated for the image that you selected in the image property when you play back the Web UI test. The UI Test report displays the test results and the correlation score between the source and the target objects. The correlation score determines if the test passed or failed. The test is passed if the correlation score is greater than or equal to the recognition threshold. Otherwise, the test is failed.

Related information

Optimizing playback of the test on page 593

Defining the Image property as object identifier for Web UI tests on page 627

Applying guided healing feature for tests identified by the image property on page 628

Defining the Image property as object identifier for Web UI tests

Before you start recording Web UI tests, you can define the image property as the object identifier. When image property is set as the object identifier, then the objects in the web application are recognized and captured as images while recording a Web UI test.

About this task

When you generate the recorded test, you can view that all objects in all the test steps are recognized and captured as images. These images are used for validating the target objects during playback of the tests. This feature helps to save time and your effort when your application has lot of images and a complex UI within your application.

1. Go to **Windows > Preferences > Test > Recording > UI Test Recording**. The **UI Test Recording** pane is displayed.
2. In the **Preferred object identifiers** pane, click **Add**.

3. Enter `image` in the text box and click **Ok**.
   Result
   Now, the image property is defined as the object identifier for Web UI tests.

Results

When you play back the Web UI tests, the target objects are identified based on the images that were captured when you recorded the Web UI test. The report displays the result based on the configuration parameters set for each image.

What to do next

You can also add the following details to your test steps:

- New images in addition to the existing images.
- Verification point to the images.

Related information

Optimizing playback of the test on page 593

Validating images and user interface elements by using the image property on page 624

Applying guided healing feature for tests identified by the image property on page 628

Applying guided healing feature for tests identified by the image property

When you play back Web UI tests that use the image property to match target with actual images, the tests fail if there is a mismatch. You can use the guided healing feature that provides the capability of finding the next best matching image so that the tests can pass.

Before you begin

You must have recorded a Web UI test. See Web UI recording on page 532.

About this task

The UI of an application might undergo changes during the development phases. When your test does not adapt to the changes in the UI elements, the probability is high that your test might fail. When you enable guided healing feature, during playback, if an object is not formally identified, then the next best matching image is selected as the possible candidate. You can choose to update the images to the test either automatically or manually.

The following procedure describes about updating the images automatically.

1. Run the recorded Web UI test from the test editor.
2. Select the **Collect data to update test steps** check box in the **Run Configuration** dialog box.
   Result
   The guided healing feature is enabled.

3. Select the **Automatically update the test at the end of playback** check box to update each test step with the identified images.
4. Click **Next**, and then click **Finish**.

   Result

   After the test run is complete, the Web UI test report is generated.

5. In the test editor pane, click **Scan and mark test with latest result**  to view the status of the test steps.

   The test steps and the images highlighted in the test steps are indicated with different colors to specify the status of each test step. You can refer to the following table to understand the meaning of each color:

Step color	Explanation	Indication of the test result
Green	During playback, the target object that matched the actual object was found based on the correlation score.	The test step is passed.
Amber	During playback, the best matching target object was not found. Then, the next closest matching image was identified as the target object based on the correlation score.	The test step is partially passed.

   Result

   During the playback, the test steps are updated with the newly identified images.

Results

The test steps are passed and displayed in black color.

Related information

Validating images and user interface elements by using the image property on page 624

Defining the Image property as object identifier for Web UI tests on page 627

Extending Web UI tests

You can perform certain tasks to extend the Web UI tests that include advanced capabilities such as adding JavaScript code to tests, replacing a JavaScript in a test script, and creating a custom Java code.

Adding custom JavaScript code as a test step in a Web UI test

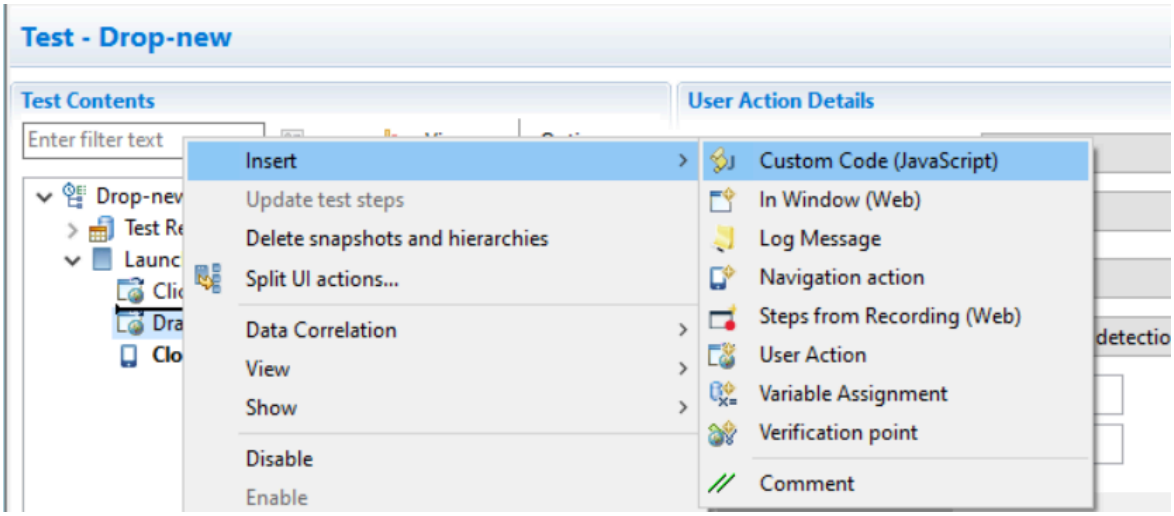
You can manually add JavaScript files (*.js) to test scripts with defined functions. You might want to run your own JavaScript snippet such as retrieving some data from the application, doing some actions within the application,

or validating some complex logic actions within the application for example. To be able to execute specific code in a test, write your own JavaScript code and insert the custom JavaScript statement as a new test step in your test script.

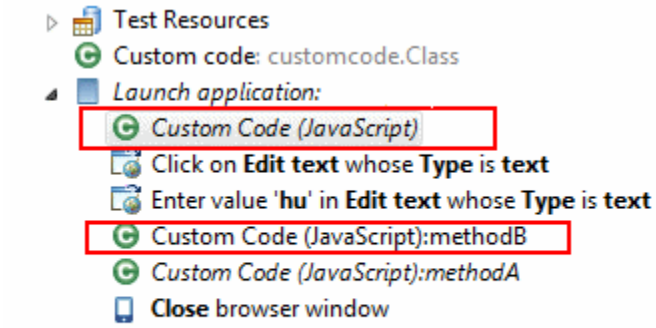
About this task

For more information on how to use the JavaScript code in the product, see the Executing JavaScript video .

1. Edit the test script.
2. Select the **Launch application** node and click on **Add** or **Insert** button in the editor and select **Custom Code (JavaScript)**, as shown in the following figure. JavaScript files with .js extension are to be kept in a project of your workspace and must be added a test step with in the launch application node.
   Figure 11. Custom Code (JavaScript) menu



3. In the dialog box that opens, select a JavaScript file to be added to the test step, click OK. It is displayed as links in Referred JavaScripts in the definition pane. A new Test Step is added to the Test script. When a method name is provided, the test step is named **Custom Code (JavaScript):method-name**, otherwise it is named **Custom Code (JavaScript)**, see figure 2.
   Figure 12. Custom code added as a test step in a Web UI test



4. Select the step to see the JavaScript custom code definition pane that contains the custom code details. Specify the JavaScript method name to be executed in the **Method** field, and optionally provide the description. Click the **Update** button to add multiple files. The JavaScript custom code will be executed within the Web application. You can also delete the referred JavaScript hyperlink, or click the link to open the JavaScript file in the editor. If the JavaScript method has some parameters to be added, specify the parameters in the **Arguments** field. You can specify the arguments through static text, a variable reference, a dataset reference, or a java custom code.
    a. To enter text values, click **Text** button and enter the text as argument.
    b. To pass test variable reference or dataset reference or JavaScript custom code return value as parameter to JavaScript method, click **Add** button. Select the available data source arguments, datasets, test variables or java custom code. The variable or dataset must be initially created, and a return value added. See example in figure 3.

Figure 13. Custom code details

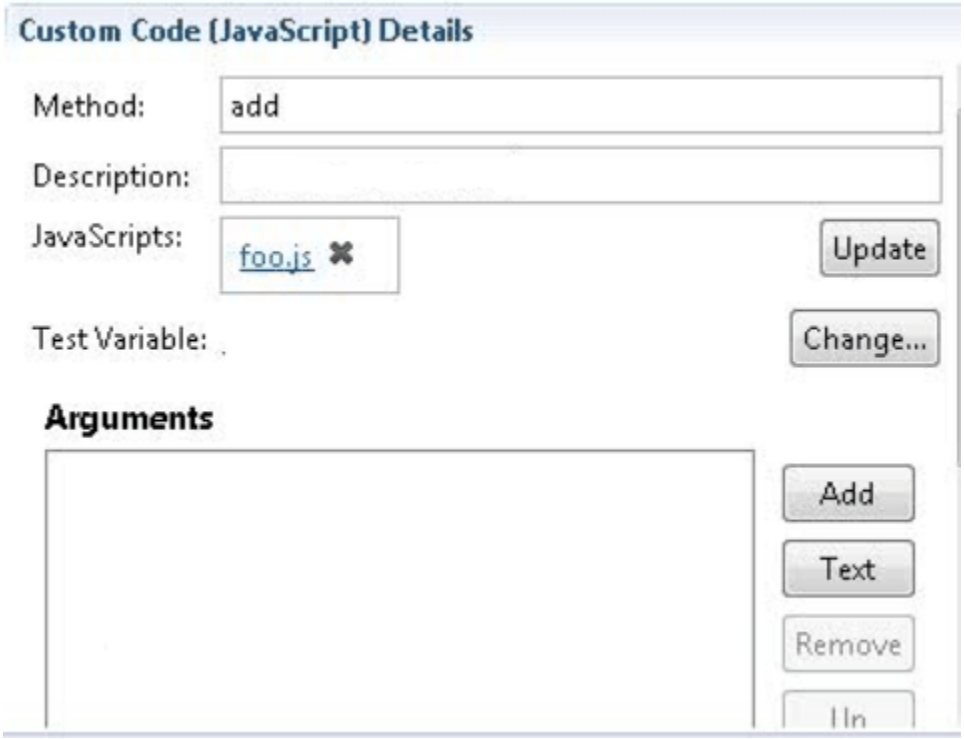
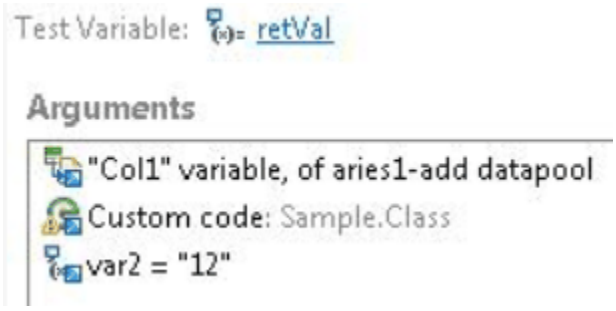


Figure 14. Example of variable and dataset as arguments



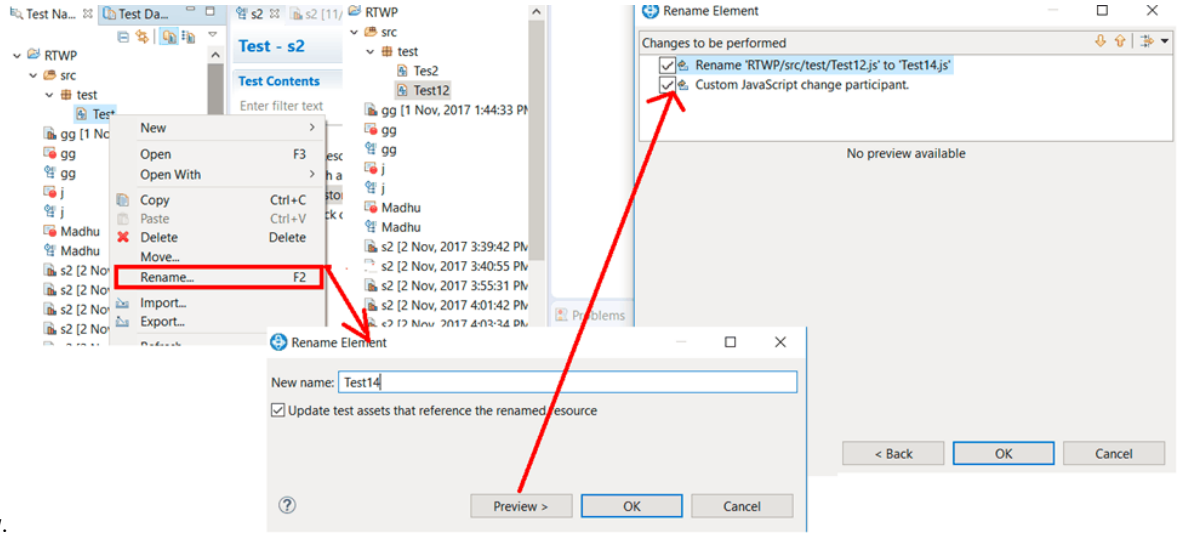
5. Run the test script and see the report.
   From version 9.1.1.1, you can replace a JavaScript file with an updated one and apply the changes to all references to the JavaScript file in the test scripts where the file is called, for more details, see Replacing a JavaScript file in a test script on page 632.

Replacing a JavaScript file in a test script

You can rename a JavaScript file in a project, delete it, or move it in the project, in all cases, the updates will apply in all test scripts in the project. When you rename a JavaScript file that is used in a Web UI test script from the Test Navigator, the system can automatically search for all references to the JavaScript file in the project and overwrite the custom JavaScript file with the updated one in all the test scripts where the JavaScript file is called. It is useful when you have multiple calls of the same JavaScript file in a single script.

To replace a JavaScript file with a new one:

a. In a project, right-click on a JavaScript file in the Test Navigator, and select **Rename.**
b. Enter a new name in the text field of the dialog that opens.
c. Select **Update test Assets that reference the renamed resource**. Click



Preview.

d. A wizard displays all references to the JavaScript files in the project. Unselect the references that should not be updated and unselect **Custom JavaScript change participant** if you don't want any changes in the associated files. Click **OK**. A message prompts you to accept or abort the changes.

Results

Once validated, the initial JavaScript file is replaced with the new JavaScript file in all the test scripts where it is called.

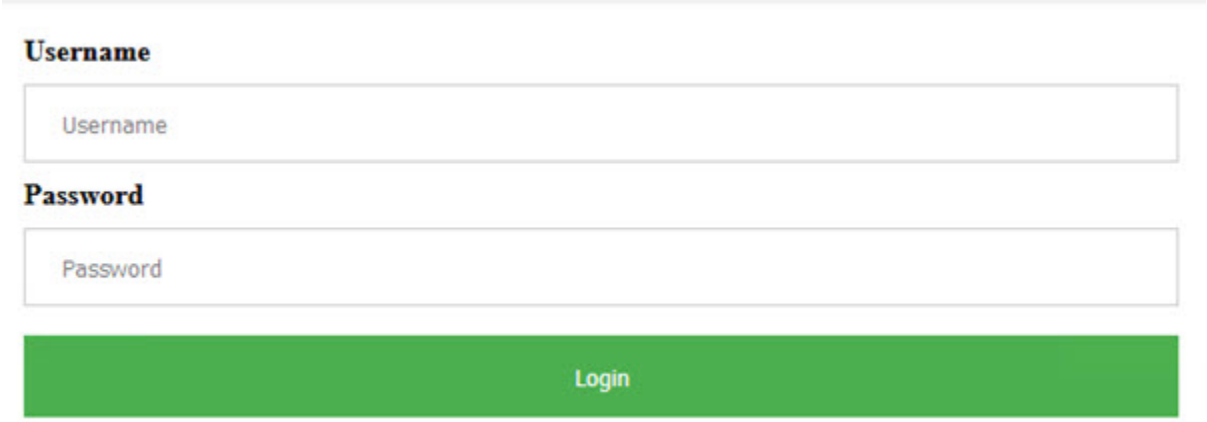
Creating a custom JavaScript code in a Web UI test

You can use the JavaScript Custom Code option to create your own code and add that as a step in the Web UI test.

This example shows you how to create your own JavaScript code and add that as a step in the Web UI test. This example covers a basic scenario of creating custom JavaScript code for a simple login form. You can create your own JavaScript code according to your functionality.

Exemple

The login form has two fields namely the Username and Password.



First, you must add a JavaScript file in the Web UI project and enter the code in the file. Refer to the following sample code created for the simple login form.

```
//Simple JS function to fill up login page data
function simpleUserLogin(){

    console.log("Signing In...");
    //Click At User Name Edit Box
    document.getElementById("uname").click();

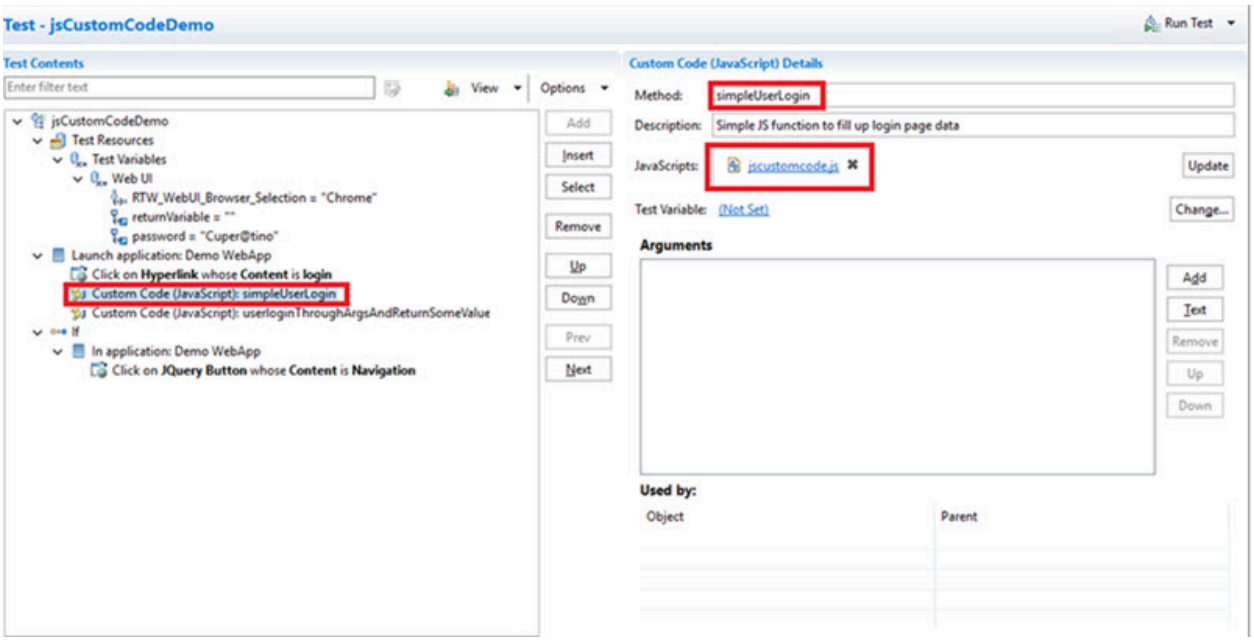
    //Enter the UserName
    document.getElementsByName("username")[0].value="billy";

    //Click on Password Field and Enter the password
    document.getElementsByClassName("form-control-passwd")[1].click();
    document.getElementById("password").value="Cuper@tino";

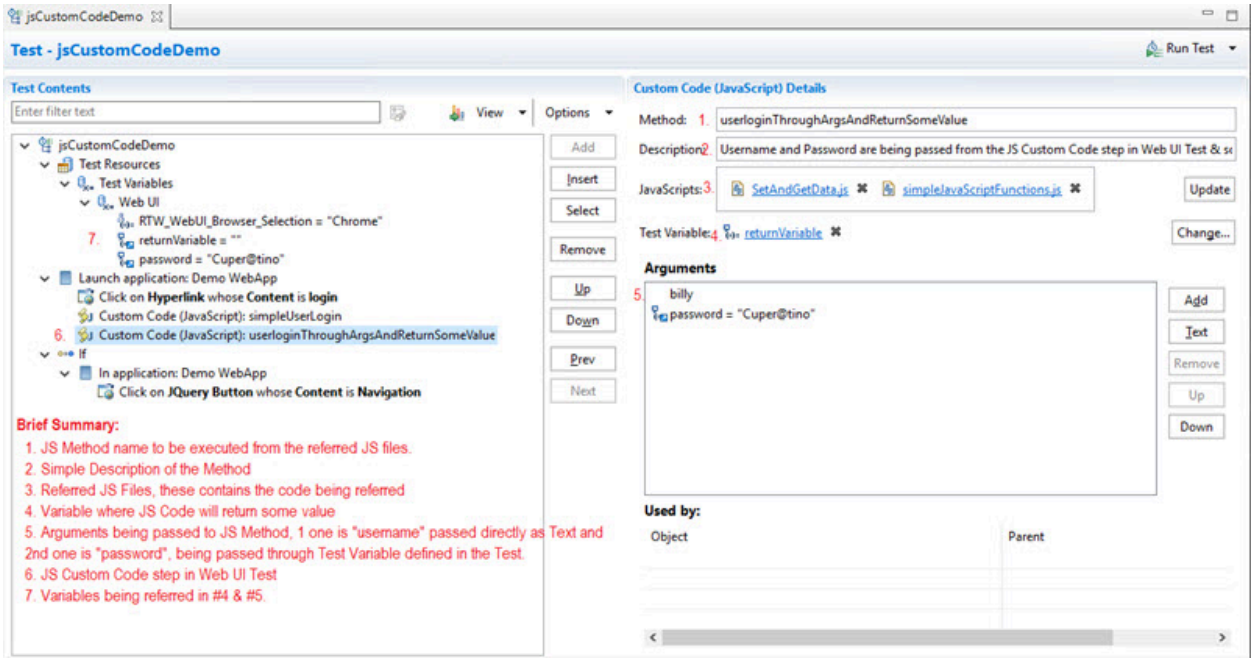
    //Click on Login button
    document.getElementsByTagName("input")[3].click();
}
```

Note: JavaScript provides you with set of libraries to interact with the DOM controls and you can use them to handle user interfaces, browsers and document settings. To know more about the JavaScript code references, see https://developer.mozilla.org/en-US/docs/Web/JavaScript.

In the Web UI project, you can update the details of the JavaScript file that you created. You have to provide the method name and update the JavaScript file as shown in the following image..



In the preceding example code, the input values such as user name and password are already hard coded. But, if you want to pass the input values from the WebUI test to the JavaScript code for which you want to get some return values, you have to create another JavaScript file and call the method in the Custom Code step as shown in the following image.



This is the sample code for defining the method that can accept values from Web UI and return values.

```
//username and password are being passed from the JS Custom Code step in Web UI Test & some value is
 being returned which would be stored in the variable defined in JS Custom Code step
function userloginThroughArgsAndReturnSomeValue(username, password){
```

```
    var returnVar = "false";
    var userName = username;
    var pwd = password;
    console.log("Waiting for the browser load...");
    //sample code - wait for document to load based on browserState
    var myVar = true;
    while(myVar === true){
        var browserState = document.readyState;
        if(browserState.indexOf("complete") !== -1){
            myVar = false;
        }
    }

    console.log("Signing In...");
    //Enter the UserName
    document.querySelectorAll("*[name='username']")[0].click();
    document.querySelectorAll("*[id='uname']")[0].value = userName;

    //Enter the password
    document.querySelectorAll(".form-control-passwd")[1].click();
    document.getElementById("password").value=pwd;

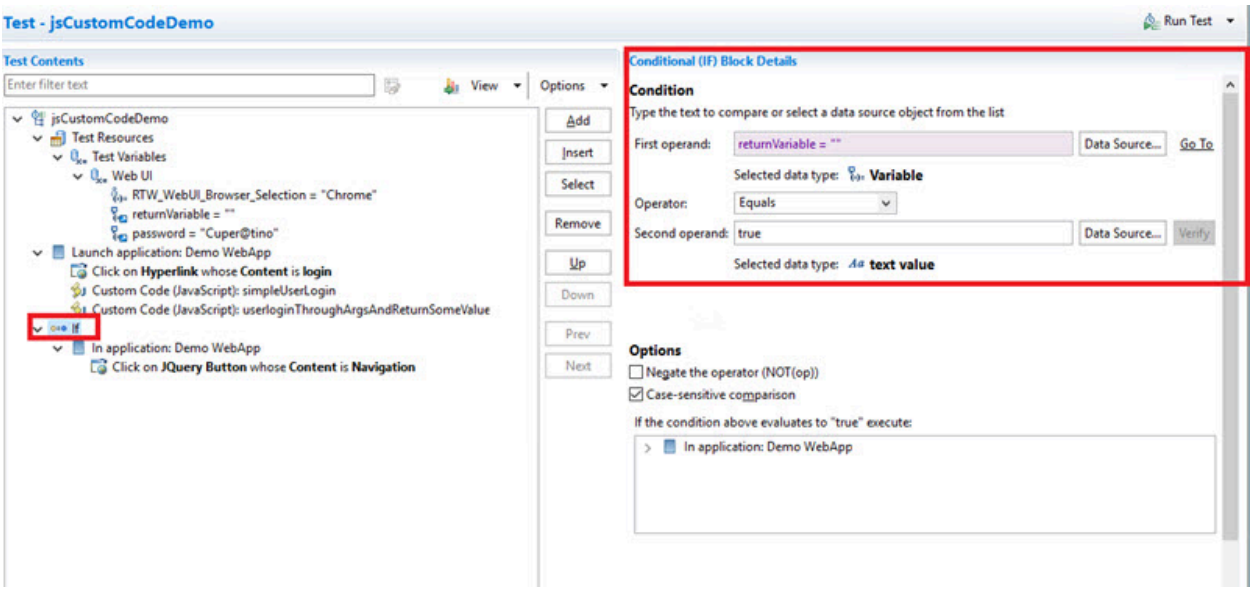
    //Click on Login button
    var submitButtonOccurances = document.querySelectorAll("*[type='submit']").length;
    document.querySelectorAll("*[type='submit']")[0].click();

    //return some value which can be stored in a variable in JS Custom Code step in Web UI Test
    console.log("Returning some value...");

    if(submitButtonOccurances>= 1){
        returnVar = "true";
    }
    return returnVar;

}
```

Based on the return value, the if condition will be executed.



Providing tests with variable data (datasets)

You can produce more realistic tests by changing them to use datasets. During execution, a test that uses a dataset replaces a value in the recorded test with variable test data that is stored in the dataset. This substitution allows each virtual user to generate a different request to the server.

Dataset overview

Datasets provide tests with variable data during a run. When you record a test, you perform a sequence of steps that you expect a typical user to perform. After the recording, a test is generated that captures these interactions. When you run this test, it uses the same data that you used during recording. To vary the data in the test, you use a *dataset*, that contains variable data. At run time, this variable data is substituted for the data in the recorded test.

If you need to create a dataset with many records, you can initialize the dataset by importing data from a comma-separated-value (CSV) file. Also, you can export test data from your dataset into a CSV file to enable you to maintain large volumes of test data as a spreadsheet for reuse. Earlier to 9.5, the dataset (formerly known as datapool) was in .datapool format and starting from the 9.5 release, the dataset is in the csv format.

You can copy the CSV file and paste into your project to import the data from a CSV file and create a dataset. Similarly, to export the dataset values as a CSV file, you must copy the dataset from your project and paste it into your local machine.

Note: Alternatively, you can use the **Import** option available in the **CSV editor** to import the data from a CSV file. For more information, see Editing datasets on page 653.

Perform the following steps should you plan to create a test that searches the IBM®HCL® website for three items: IBM® Rational® Performance TesterHCL OneTest™ Performance, IBM® Rational® Functional TesterHCL OneTest™ UI, and IBM® Rational® Manual Tester:

1. Record a test that searches for one item.
2. Create a dataset and associate it with the test. For more information, see Creating a dataset associated with a test on page 637.
3. Associate a request in the test with a column in the dataset. For more information, see Associating a test value with a dataset column on page 651.
4. Add a loop in the test to fetch the values from different rows of a dataset. A test without a loop fetches the value only from the first row of the dataset.

Creating a dataset associated with a test

You can create a dataset that contains variable data for tests to use when they run. This is the preferred way to create a dataset because the dataset is automatically associated with a test. You can create anything from an empty dataset that contains one column, which you can edit later, to a fully functioning dataset.

1. In the Test Navigator, browse to the test and double-click it.
   Result
   The test opens.
2. In the **Test Contents** area, click the name of the test.
3. In the **Common Options** tab, click **Add Dataset**.
   The options listed in the following table, enable you to create anything from a simple dataset that you can edit later to a complete dataset.

To create	Do this in the Test Editor - Add Dataset window
A one-column dataset with a default access mode.	In **Existing datasets in workspace**, select *New Dataset<testname>.csv*, and click **Finish**. You can optionally name the dataset column in this session, and you can add other columns and data later.
A one-column dataset and choose the access mode.	In **Existing datasets in workspace**, select *New Dataset<testname>.csv*, and click **Next**. You can optionally name the dataset column in this session and you are prompted for the access mode. You can add other columns and data later.
An association between the test and an existing dataset.	Select the dataset. The dataset is associated with the test, and you can optionally set the access mode in this session.
A new, fully functioning dataset.	Select a project and click **Use wizard to create new dataset**.

4. Select **Open mode** for the dataset. This mode determines the view that virtual users have of the dataset. Different tests can open the same dataset differently, and you can change the open mode later by opening the test and double-clicking the dataset title.

Option	Description
Shared (per test execution) (default)	When you choose the Shared (per test execution) option, the virtual users running in the test share the dataset values in sequential order. For example, if your dataset has 10 rows, the dataset values are taken from row 1, row 2, row 3, and so on when you select this option.
Private	Each virtual user draws dataset values from a private view of the dataset, with dataset rows apportioned to each user in sequential order. This option ensures that each virtual user gets the same data from the dataset in the same order. Because each user starts with the first row of the dataset and accesses the rows in order, different virtual users will use the same row. The next row of the dataset is used only if you add the test that is using the dataset to a loop in the schedule with multiple iterations.
Shared (for all test executions)	When you choose the Shared (for all test executions) option, the virtual users running in multiple tests share the dataset values from the current row. For example, if your dataset has 10 rows and when you set the current row as row 5, the dataset values are taken from row 5 instead of row 1 when you select this option. If you had set the current row as row 1 and used the dataset values until row 5, the dataset values are retrieved from row 6 when you run the test next time.

5. If you are setting how the test accesses the dataset during this session, select one of the following options:

- **Sequential**: Rows in the dataset are accessed in the order in which they are physically stored in the dataset file, beginning with the first row and ending with the last.
- **Random**: Rows in the dataset are accessed in any order, and any given row can be accessed multiple times or not at all. Each row has an equal chance of being selected each time.
- **Shuffled**: Before each dataset access, the order of the rows is changed that results in a different sequence. The rows are accessed randomly but all rows must be selected once before a row is selected again.

6. Select one of the following options.

Option	Description
Wrap when the last row is reached	By default, when a test reaches the end of a dataset or dataset segment, it reuses the data from the beginning. To force a test to stop at the end of a dataset or segment, clear the **Wrap when the last row is reached** check box. Forcing a stop might be useful if, for example, a dataset 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 reusing dataset data does not matter to your application, the default of wrapping is more convenient. With wrapping, you need not ensure that your dataset is large enough when you change the workload by adding more users or increasing the iteration count in a loop. **Note:** ◦ With **Random** access order, **Wrap when the last row is reached** option is unavailable because you never reach the end of the row. ◦ With **Shuffled** access order, if you select **Wrap when the last row is reached** option, you resume selecting from the beginning of the row with the same access order after each row has been selected once. No more se-

Option	Description
	 lections are required if you clear the **Wrap when the last row is reached** option.
Fetch only once per user	By default, one row is retrieved from the dataset for executing each test, and the data in the dataset row is available to the test only for the duration of the test. Select **Fetch only once per user** to specify that every access of the dataset from any test being run by a particular virtual user will always return the same row.

Example

To illustrate how these options affect the rows that are returned, assume that a test contains a loop which accesses a dataset. The loop has 2 iterations. The following table shows the row that is accessed in each iteration:

Dataset option	Iteration 1	Iteration 2
Sequential and **Private**	row 1	row 2
Shared and **Shuffled**	row x	row y
Fetch only once per user	row x	row x

7. If you are creating a fully functioning dataset, you can optionally import the data from a CSV file during this session by copying the CSV file and pasting into your project. For more information on importing dataset, see Editing datasets on page 653.

What to do next

After you have created a dataset and added data to it, the next step is to associate a value in the test with a column in the dataset, as discussed in Associating a test value with a dataset column on page 651.

Creating a dataset in a workspace

You can create datasets in a workspace containing variable data that tests use when they run. You can use this method to create a dataset if you have not yet created the test that will use it.

1. Click **File > New > Dataset**.
2. In the **New Dataset** window, click the project that contains the dataset. The project is displayed in the **Enter, create, or select the parent folder** field.
3. In the **Name** field, type the name of the dataset, and then click **Next**.

4. In the window for describing the dataset, add a description.
5. In the **Dimensions** field, specify the number of rows and columns for the dataset that you want to create.
6. Click **Finish**.

Results

The new dataset opens in a browser. For instructions on how to add data to or edit the dataset, see Editing a dataset on page 653.

What to do next

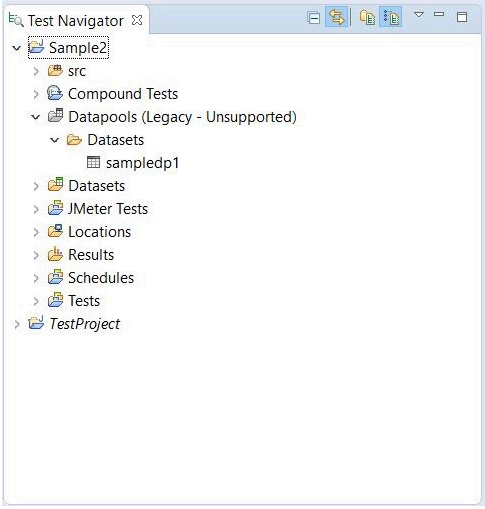
After you have created a dataset and added data to it, you must associate a value in the test with a column in the dataset.

Converting an existing datapool to a dataset

Starting from 9.5 the dataset formerly known as datapool is in the CSV format. You can convert any existing datapool to a dataset.

About this task

When you open the workspace earlier to 9.5 in IBM® Rational® Performance TesterHCL OneTest™ Performance 9.5, the existing datapools in the workspace are stored in the Datapools (Legacy-Unsupported) folder as shown in the following figure.



To convert the existing datapool to a dataset:

1. In the Test Navigator, browse and select the existing datapool.
2. Right-click and select **Convert to Dataset...**. Verify that the name of the dataset is the name of the existing datapool and format is .csv.
3. Click **Finish**. The converted datapool opens in a CSV editor.

What to do next

After you have created a dataset and added data to it, you must associate a value in the test with a column in the dataset.

Creating datasets with multiple substitutions

Earlier to 9.2, you could substitute one dataset value at a time. Starting from 9.2, after the test is generated, you can view all the dataset candidates, add multiple candidates as dataset values, substitute values, and create a new dataset out of it. You can also substitute multiple dataset candidates for an existing dataset.

About this task

When you substitute multiple dataset candidates to create a new dataset, the same number of columns are created in the dataset. The names of the candidates become the names of columns and values in the dataset. When you substitute multiple dataset candidates in an existing dataset, the column names in the dataset are retained. If the number of substitutions chosen was greater than the number of columns in the dataset, the extra number of substitutions are added as columns in the dataset. For instance, if a dataset has three columns and you substitute five dataset candidates, two new columns are created by using the names of the dataset candidates.

To create a dataset from multiple dataset candidates:

1. In the Test Editor, select the name of the test and from the Test Details section, select **Common Options** and click **Show Dataset Candidates**.
   Alternative: After the test generation when you open the test, you are prompted that *"Some test data may need to be correlated or substituted"*. If you click **Yes**, you can see the list of dataset candidates.
2. Select the dataset candidates that you want to add as values to the dataset and click **Substitute multiple candidates**.
   The **Add Dataset** dialog box shows the list of datasets that are in the project but not associated with the test.
3. To associate an existing dataset with the test and assign the selected dataset candidates as values and substitutions, select a dataset and click **Next**. To associate a new dataset with the test, click the **Use wizard to create new Dataset** and click **Next**.
4. Select **Open mode** for the dataset. This mode determines the view that virtual users have of the dataset. Different tests can open the same dataset differently, and you can change the open mode later by opening the test and double-clicking the dataset title.

Option	Description
Shared (per test execution) (default)	When you choose the Shared (per test execution) option, the virtual users running in the test share the dataset values in sequential order. For example, if your dataset has 10 rows, the dataset values are taken from row 1, row 2, row 3, and so on when you select this option.

Option	Description
Private	Each virtual user draws dataset values from a private view of the dataset, with dataset rows apportioned to each user in sequential order. This option ensures that each virtual user gets the same data from the dataset in the same order. Because each user starts with the first row of the dataset and accesses the rows in order, different virtual users will use the same row. The next row of the dataset is used only if you add the test that is using the dataset to a loop in the schedule with multiple iterations.
Shared (for all test executions)	When you choose the Shared (for all test executions) option, the virtual users running in multiple tests share the dataset values from the current row. For example, if your dataset has 10 rows and when you set the current row as row 5, the dataset values are taken from row 5 instead of row 1 when you select this option. If you had set the current row as row 1 and used the dataset values until row 5, the dataset values are retrieved from row 6 when you run the test next time.

5. If you are setting how the test accesses the dataset during this session, select one of the following options:
    - **Sequential**: Rows in the dataset are accessed in the order in which they are physically stored in the dataset file, beginning with the first row and ending with the last.
    - **Random**: Rows in the dataset are accessed in any order, and any given row can be accessed multiple times or not at all. Each row has an equal chance of being selected each time.
    - **Shuffled**: Before each dataset access, the order of the rows is changed that results in a different sequence. The rows are accessed randomly but all rows must be selected once before a row is selected again.
6. Select one of the following options.

Option	Description
Wrap when the last row is reached	By default, when a test reaches the end of a dataset or dataset segment, it reuses the data from the beginning. To force a test to stop at the end of a

Option	Description
	dataset or segment, clear the **Wrap when the last row is reached** check box. Forcing a stop might be useful if, for example, a dataset 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 reusing dataset data does not matter to your application, the default of wrapping is more convenient. With wrapping, you need not ensure that your dataset is large enough when you change the workload by adding more users or increasing the iteration count in a loop. **Note:** ◦ With **Random** access order, **Wrap when the last row is reached** option is unavailable because you never reach the end of the row. ◦ With **Shuffled** access order, if you select **Wrap when the last row is reached** option, you resume selecting from the beginning of the row with the same access order after each row has been selected once. No more selections are required if you clear the **Wrap when the last row is reached** option.
Fetch only once per user	By default, one row is retrieved from the dataset for executing each test, and the data in the dataset row is available to the test only for the duration of the test. Select **Fetch only once per user** to specify that every access of the dataset from any test being run by a particular virtual user will always return the same row.

Example

To illustrate how these options affect the rows that are returned, assume that a test contains a loop which accesses a dataset. The loop has 2 iterations. The following table shows the row that is accessed in each iteration:

Dataset option	Iteration 1	Iteration 2
Sequential and **Private**	row 1	row 2
Shared and **Shuffled**	row x	row y
Fetch only once per user	row x	row x

7. Click **Finish**.

Adding Dataset Mapper

You can include a Dataset Mapper in a compound test or a schedule to assign the dataset values to the variables that are defined in multiple tests. In previous releases, to apply the dataset values to multiple tests, you had to associate the dataset to each test. The Dataset Mapper is able to map the dataset columns with the variables.

Before you begin

You must have created at least one dataset. See Creating a dataset in a workspace on page 640.

About this task

For the Dataset Mapper to fetch the test variables, in the Variable Details section of the Test editor, you must set the **Visible In** field for the variable to **All tests for this user**. You can also fetch the variables from the custom code calls.

If the compound test or the schedule includes a Dataset Mapper that retrieves values from one dataset and a test in the compound test or schedule is also associated with another dataset, the run uses both the datasets.

Note: When you run the schedule or compound test with a Dataset Mapper, by default the test picks up the dataset values from the first row. For the test to pick up all of the dataset values, you must put the test in a loop.

1. In the Schedule or compound test editor, click **Add > Dataset Mapper**.
2. In the **Select Dataset** dialog box, select a dataset to use for the tests and click **OK**.
   To change the dataset after it is associated, in **Dataset Mapper Details**, click **Browse** and select another dataset.
3. Select the **Open mode** for the dataset. This mode determines the view that virtual users have of the dataset. This option is useful when you do a parallel test run.

Option	Description
Shared (per test execution) (default)	When you choose the Shared (per test execution) option, the virtual users running in the test share the dataset values in sequential order. For example, if your dataset has 10 rows, the dataset values are taken from row 1, row 2, row 3, and so on when you select this option.
Private	Virtual users draw from a private view of the dataset, with dataset rows apportioned to each user in sequential order. This option ensures that each virtual user gets the same data from the dataset in the same order. However, because each user starts with the first row of the dataset and accesses the rows in order, different virtual users will use the same row. The next row of the dataset is used only if you add the test that is using the dataset in a loop with more than one iteration.
Shared (for all test executions)	When you choose the Shared (for all test executions) option, the virtual users running in multiple tests share the dataset values from the current row. For example, if your dataset has 10 rows and when you set the current row as row 5, the dataset values are taken from row 5 instead of row 1 when you select this option. If you had set the current row as row 1 and used the dataset values until row 5, the dataset values are retrieved from row 6 when you run the test next time.

4. Select the **Access mode** for the dataset:

- **Sequential**: Rows in the dataset are accessed in the order in which they are physically stored in the dataset file, beginning with the first row and ending with the last.
- **Random**: Rows in the dataset are accessed in any order, and any given row can be accessed multiple times or not at all. Each row has an equal chance of being selected each time.
- **Shuffled**: Before each dataset access, the order of the rows is changed that results in a different sequence. The rows are accessed randomly but all rows must be selected once before a row is selected again.

5. Select whether the test will reuse data when it reaches the end of the dataset.

   By default, when a test reaches the end of a dataset or dataset segment, it reuses the data from the beginning. To force a test to stop at the end of a dataset or segment, clear the check box **Wrap when the last row is reached**. Forcing a stop might be useful if, for example, a dataset 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 as *"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 dataset is large enough when you change the workload by adding more users or increasing the iteration count in a loop.

6. Select whether the test will make the data in the dataset record permanent for each virtual user.

   By default, one row is retrieved from the dataset for each execution of a test, and the data in the dataset row is available to the test only for the duration of the test. Select **Fetch only once per user** to specify that every access of the dataset from any test being run by a particular virtual user will always return the same row.

   To illustrate how these options affect the rows that are returned, assume that a test contains a loop which accesses a dataset. The loop has two iterations. The following table shows the row that is accessed in each iteration:

Dataset option	Iteration 1	Iteration 2
Sequential and **Private**	row 1	row 2
Shared and **Shuffled**	row x	row y
Fetch only once per user	row x	row x

7. In the **Columns mapping** table, the **Column** is automatically filled with the column names from the dataset.
8. To use the variable names from the test, click the cell and click the Ellipsis button  and select the variable. By default, the variable names are also created with the same names as the dataset columns.
9. If the dataset that you selected in step 2 was generated by HCL® OneTest™ Data, you can choose to update the data by clicking **Update dataset** or update the data automatically for every run by selecting the **Update dataset during deployment** check box.
10. To fetch all the dataset values, put the Dataset Mapper in a loop. Select the **Dadtapool Mapper** in the schedule and click **Insert > Loop**.
11. Save the changes.

How dataset options affect values that a virtual user retrieves

The Open, Access, and Wrap modes that you select for a dataset affect the values that a virtual user retrieves.

The following table lists the most common types of datasets and the options that you select to create them.

Dataset purpose	Open mode selection	Access mode selection	Wrap mode selection
The virtual user retrieves the value from the current row of the dataset in a random order for every attempted transaction. Note that before accessing each row of the dataset the order of the rows is rearranged.	Shared (for all test executions)	Shuffled	Fetch only once per user
The virtual user retrieves the value from the current row of the dataset in sequential order for every attempted transaction.	Shared (for all test executions)	Sequential	Fetch only once per user
The virtual user retrieves the value from the beginning of the row of a dataset in a random order for every attempted transaction.	Shared (per test execution)	Random	Wrap when the last row is reached
The virtual user retrieves the value from the current row of a dataset in sequential order for every attempted transaction. When a test reaches the end of a dataset, it reuses the data from the current row selection of the dataset.	Shared (for all test executions)	Sequential	Wrap when the last row is reached

Enabling a test to use a dataset

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

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. Right-click the test name, and click **Add > Dataset**.
   Result
   The **Select Dataset File** window is displayed listing the datasets available to the test. If a test is already using a dataset, it does not appear in the list.
3. In the **Existing Dataset in workspace** list, click the name of the dataset that your test will use and click **Next**.
4. Select the **Open mode** for the dataset. This mode determines the view that virtual users have of the dataset. This option is useful when you do a parallel test run.

Option	Description
Shared (per test execution) (default)	When you choose the Shared (per test execution) option, the virtual users running in the test share the dataset values in sequential order. For example, if your dataset has 10 rows, the dataset values are taken from row 1, row 2, row 3, and so on when you select this option.
Private	Virtual users draw from a private view of the dataset, with dataset rows apportioned to each user in sequential order. This option ensures that each virtual user gets the same data from the dataset in the same order. However, because each user starts with the first row of the dataset and accesses the rows in order, different virtual users will use the same row. The next row of the dataset is used only if you add the test that is using the dataset in a loop with more than one iteration.
Shared (for all test executions)	When you choose the Shared (for all test executions) option, the virtual users running in multiple tests share the dataset values from the current row. For example, if your dataset has 10 rows and when you set the current row as row 5, the dataset values are taken from row 5 instead of row 1 when you select this option. If you had set the current row as row 1 and used the dataset values until row 5, the dataset values are retrieved from row 6 when you run the test next time.

5. Select the **Access mode** for the dataset:

- **Sequential**: Rows in the dataset are accessed in the order in which they are physically stored in the dataset file, beginning with the first row and ending with the last.
- **Random**: Rows in the dataset are accessed in any order, and any given row can be accessed multiple times or not at all. Each row has an equal chance of being selected each time.
- **Shuffled**: Before each dataset access, the order of the rows is changed that results in a different sequence. The rows are accessed randomly but all rows must be selected once before a row is selected again.

6. Select whether the test will reuse data when it reaches the end of the dataset.

   By default, when a test reaches the end of a dataset or dataset segment, it reuses the data from the beginning. To force a test to stop at the end of a dataset or segment, clear the check box **Wrap when the last row is reached**. Forcing a stop might be useful if, for example, a dataset 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 as *"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 dataset is large enough when you change the workload by adding more users or increasing the iteration count in a loop.

7. Select whether the test will make the data in the dataset record permanent for each virtual user.

   By default, one row is retrieved from the dataset for each execution of a test, and the data in the dataset row is available to the test only for the duration of the test. Select **Fetch only once per user** to specify that every access of the dataset from any test being run by a particular virtual user will always return the same row.

   To illustrate how these options affect the rows that are returned, assume that a test contains a loop which accesses a dataset. The loop has two iterations. The following table shows the row that is accessed in each iteration:

Dataset option	Iteration 1	Iteration 2
Sequential and **Private**	row 1	row 2
Shared and **Shuffled**	row x	row y
Fetch only once per user	row x	row x

8. Click **Finish**.

   Result

   A reference to the dataset is added to the test, and the **Test Details** area is updated with the dataset information.

9. Save the test.

What to do next

Now that you have created a reference between the test and the dataset, the next step is to associate a value in the test with a column in the dataset.

Associating a test value with a dataset column

After you have created a dataset and have enabled your test to use the dataset, you can associate a specific value in the test with a specific dataset column.

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. Locate and click a request that contains a value to replace with variable data.

   Clicking a test page displays a table that lists dataset candidates and correlated data on that page. (If correlated data is not displayed, right-click the table and verify that **Show References** is selected.) References are color coded in blue and dataset candidates are color coded in black.

   Test Data

Name	Value	Substituted with
searchFor	doe%2C+john	

   If the contents of the Value column corresponds exactly with column data in your dataset, click the row, and then click **Substitute**. The **Select Data Source** window is displayed. Skip to step 6. You can ignore step 8 because the URL encoding is preselected.

   Otherwise, double-click the row to navigate to the page request that contains the value to replace from a dataset, and continue to the next step.

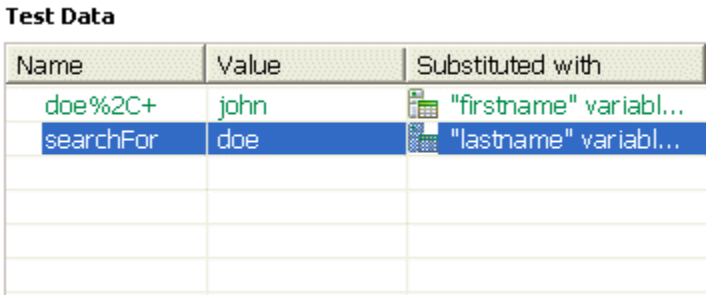
   The value to replace from a dataset might not be listed in any page table. In this case, manually locate the request string that includes the value.
3. If the value to replace from a dataset is part of a string that has been designated a dataset candidate, you must remove the light green highlight: right-click and select **Remove Substitution**.
   For example, if you searched for **doe, john** in your test, the dataset candidate in your test is displayed as **doe%2C+john**. Suppose that you do not want to associate this candidate with a single dataset column that contains data in the format **doe, john**. Instead, you want to associate **doe** and **john** with separate dataset columns. In this case, you must first remove the substitution.
4. Highlight the value: With the left button pressed, drag your mouse over the value.
5. Right-click the highlighted value, and select **Substitute > Select Data Source**.
   Result

   The **Select Data Source** window is displayed.

   Note: To use a dataset that is not listed, click **Dataset**: the **Select dataset column** window is displayed.
6. Click the name of the dataset variable, or column, to associate with the test value.
7. Click **Select**.

Result

To indicate that the association has been set, the highlighting for the selected test value turns dark green, and the dataset table for this page is updated as shown in the example.



8. **Optional:** Encode variable data when it is substituted from a dataset.

   If a test value contains special characters such as spaces or commas, click the row and select **URL Encode**. With this option, special characters are encoded when variable data is substituted from a dataset. For example, data that is space-separated in a dataset column might need to be encoded. When the URL encoding is enabled, **John Doe** is substituted as **John%20Doe**. If the URL encoding is not selected, the variable data that is substituted is literal. Do not enable URL encoding for datasets that contain data that is already encoded.
9. **Optional:** If you substitute an element of a page with a dataset column, to view the substitutions in the Page Elements report, in the Test Elements Details area of the request click the **Use the substituted URL in performance reports** check box.
10. Save the test.

Related information

Adding data source controller

Viewing dataset candidates when you open a test

Dataset candidates are displayed automatically when you open a test for the first time. From the dataset candidates window you can view the dataset candidates in the test, bookmark locations of interest, and add or remove dataset references.

1. Record a test.

   Result

   When the test opens for the first time in the Test Navigator, the **Show Dataset Candidates** window is displayed. The **Show Dataset Candidates** window is displayed only if there are dataset candidates and if **Always display this dialog when a test is first opened** is selected. To prevent the **Show Dataset Candidates** from being displayed when a test opens, clear the **Always display this dialog when a test is first opened** check box in the **Show Dataset Candidates** window.
2. Do one of the following:

Option	Description
To view details about the dataset candidates in a test	Navigate through the **Dataset Candidates** field to see them previewed in the **Preview** pane. Click the **Next** and **Previous** icons to move the selection down or up in the list of dataset candidates. Click the **Show as Tree** icon to toggle between tree format and list format. Click the **Sort** icon to sort the list of dataset candidates. Click the **Bookmark** icon to bookmark a location for later review.
To select a data source for a dataset candidate	Select the dataset candidate in the **Dataset Candidates** field, and then click **Substitute**. The **Select Data Source** window opens.
To find more values in the test that have the same value as the selected dataset candidate	Click **Find More and Substitute**. These values can be reviewed and substituted interactively as needed.
To remove a substitution	Select a substitution site, and then click **Remove Substitution**.

3. Click **Close** to close the **Show Dataset Candidates** window and proceed to the test in the test editor.
   To display the **Show Dataset Candidates** window again while in the test editor, click the root node of the test. Then click the **Common Options** tab under **Test Element Details**, and then click **Show Dataset Candidates**.

Editing datasets

You can add, modify, remove, import, or export data from a dataset by using the CSV Editor. The working principle of the CSV Editor is similar to that of a spreadsheet.

Before you begin

You must have created a dataset. See Creating a dataset in a workspace on page 640.

About this task

In Rational® Functional TesterHCL OneTest™ UI 9.5.0 or later, you can use the CSV Editor to view and edit data in the dataset. You can also view the datasets in other editors by right-clicking the dataset and selecting the **Open With** option.

You can perform basic tasks in the CSV Editor by right-clicking any row, column, or cell of the dataset to organize your data in a better way. For example, updating the data in a cell, inserting or deleting rows and columns, or renaming column names.

When you edit the dataset in a CSV Editor, you can use the following keyboard shortcuts to control the cursor selection in the CSV Editor:

- **Tab** - To move the cursor control to the next available option.
- **Shift-Tab** – To move the cursor control to the previous option.
- **Shift+F10** – To open the context menu from the dataset cell.

Note: You cannot resize the width of rows in the CSV Editor. When you have a large amount of data in a cell, you can right-click the cell and select **Copy** (or Ctrl+C), and then paste it into a text-editing program to view the content. Alternatively, you can hover the mouse over the cell to view the content.

When you have a CSV file that has data separated from a character, then you can import that CSV file into the dataset. You can select any of the following separator characters from the **Configure Dataset** window, and the selection can be the separator character that you used in the CSV file:

- Comma
- Semicolon
- Space
- Tab
- Other

Consider that you have the data in the CSV file in the following format:

Name;CCNum
John;1234 5678 1234 5678
Bob;1122 3344 5566 7788
Amy;2233 4455 6677 8899

When you import the CSV file in the dataset, and then select the separator value as **Semicolon**, the data in the dataset is displayed as follows:

	Name	CCNum
1	John	1234 5678 1234 5678
2	Bob	1122 3344 5566 7788
3	Amy	2233 4455 6677 8899

If you want the data in its original format, that is, a semicolon (;) character to separate the data, then you can choose any other separator value from the **Configure Dataset** window.

Note: The default separator value is **Comma**.

1. Double-click the dataset that you want to edit in the **Test Navigator**.
   Result
   The dataset opens in the CSV Editor in a browser.
2. Perform the following actions to use the options available in the CSV Editor:

Options	Actions
Find and Replace	To find: a. Click the **Find and Replace** icon . b. Enter the content that you want to search in the **Find** field. c. Select any or all the following options to find the search content more effectively: ▪ Select the **Case sensitive** check box to search the content that is the exact letter case of the content entered in the **Find** field. ▪ Select the **Match entire cell contents** check box to search for cells that contain only the characters that you have entered in the **Find** field. ▪ Select the **Search using regular expression** check box to search the pattern that matches strings. For example, to search a cell that contains any number between 0 to 9, do the following: i. Enter *d* in the **Find** field. ii. Select the **Search using regular expression** check box. iii. Click **Find**. d. Click **Find**. If the text is found, the cell containing that text is selected. e. Click **Find** again to find further instances of the search text. To find and replace: a. Click the **Find and Replace** icon . b. Enter the content that you want to search in the **Find** field. c. Enter the content that you want to replace in the **Replace** field. d. Select any or all the following options to find and replace the content more effectively: ▪ Select the **Case sensitive** check box to find the content that is the exact letter case of the content entered in the **Find** field. ▪ Select the **Match entire cell contents** check box to find and replace for cells that contain only the characters that you have entered in the **Find** and **Replace** fields. ▪ Select the **Search using regular expression** check box to find and replace the pattern that matches strings.

Options	Actions
	e. Click **Replace** to replace the individual instances. f. Click **Replace All** to replace every instance of the content throughout the dataset.
Undo	a. Click the **Undo** icon . b. Select the recent changes from the list that you want to undo, and then click the list. The **Undo** option undoes anything you do in the dataset. The CSV Editor saves the unlimited undo-able action. You can perform the undo action even after you save your changes made to the dataset.
Redo	a. Click the **Redo** icon . b. Select the recent changes from the list that you want to redo, and then click the list. The CSV Editor saves the unlimited redo action.
Import	a. Click the **Import** icon . b. Click **Choose File** and select the CSV file that you want to import in the **Import File** dialog box. **Note:** If the CSV file contains test data with Unicode characters in it, you must save the CSV file in UTF-8 format. You can then choose the CSV file and import the test data from the CSV file into the dataset. c. Optional. Click **Overwrite** to add the rows and columns from the selected CSV file from the beginning of the dataset. d. Optional. Click **Append** to add rows and columns from the selected CSV file to the end of the dataset. e. Optional. Select the **First row contains headers** check box if your CSV file contains the header.
Export	Click the **Export** icon to download the dataset as a CSV file.
Set as current row	Right-click any cell in a row and select **Set as current row**. **When rows are deleted:** If you delete any row between row 1 to current row, the current row data is taken from the next row. For example, when you set the current row as 6, and then you delete any row between row 1 to row 6, the current row remains at row 6, but the content of row 7 is moved to row 6. **When rows are inserted:**

Options	Actions
	If you insert any new row between row 1 to the current row, the current row data is taken from the previous row. For example, when you set the current row as 6, and then you insert any row between row 1 to row 6, the current row remains at row 6, but the content of row 5 is moved to row 6.
Dataset configuration settings 	In the **Configure Dataset** window, you can set the separator value, change the row and column settings, and configure the string values in the dataset. a. Click the **Menu** icon , and then select the **Configure** option. b. Select any of the separator values that you used in the CSV file. The available options are **Comma**, **Semicolon**, **Space**, **Tab**, and **Other**. In the CSV file, if you have any other separator characters other than the available options, then you can select the **Other** option, and then can specify a value. For example, if the data in the CSV file is separated by a character #, then select the **Other** option and enter # in the field. c. Configure the following options to change the row and column settings: ▪ **Column header** - Use an up-down control button to increment or decrement the value of the column header. ▪ **Data start point** - Use an up-down control button to increment or decrement the value of the data starting pointer. ▪ **Current row** - Use an up-down control button to increment or decrement the value of the current row. d. Configure the following options to change the string values in the dataset: ▪ **Treat as null** - Enter a string value that is to be treated as null when running the test. ▪ **Treat as empty** - Enter a string value that is to be treated as empty when running the test.

Options	Actions
	For example, when you run the test and the data *123* in the dataset to be treated as empty, then you can specify *123* in the **Treat as empty** field. ▪ **Treat empty text as null** - Select this field when you want the dataset that contains any blank cells, and the value of those blank cells to be interpreted as null. e. Click **Update** to apply the changes.
Discard 	Click the **Menu** icon , and then select **Discard** to discard the changes made to the dataset.

3. Click the **Save** icon  to save the changes made to the dataset.

Results

You have edited the dataset.

Encrypted datasets overview

You can encrypt one or more columns in a dataset. If you want to encrypt confidential information such as a set of passwords or account numbers that are used during a test, you can use an encrypted dataset.

Dataset columns are encrypted using the RC4 private-key algorithm. You can use only one password to encrypt columns in any given dataset. Encrypted datasets are not supported on agent computers that are running the z/OS® or AIX® operating systems.

 **Important:** If you forget the password to a dataset, there is no way to recover the password.

When you run a test that uses a dataset that contains encrypted variables, you are prompted for the dataset password. If the test uses multiple encrypted datasets, you must enter the password for every encrypted dataset that the test uses.

When you run a test that uses a dataset with an encrypted column, the value of the column is decrypted at a run time. The column value is sent as a cleartext string in the requests to the server. The actual values of the encrypted dataset variables are not displayed in the test log. The test log displays asterisks for the encrypted dataset variables.

To see the actual values of variables that are sent to the server at run time, you must use custom code. You can send the dataset column value to custom code that writes the value to a file other than the test log. If the custom code writes to the test log using `tes.getTestLogManager().reportMessage()`, then asterisks are displayed instead of the decrypted variables.

Encrypting a dataset column

To secure test data, you must encrypt datasets. You can encrypt data in the columns of a dataset by using an encryption key. When you run a test that utilizes a dataset with encrypted variables, you must enter the encryption key for the encrypted column that the test uses.

Before you begin

You must have created a dataset. See Creating a dataset in a workspace on page 640.

1. Double-click the dataset in the Test Navigator.
   Result
   The dataset is displayed in a browser.
2. Right-click any cell in a column that you want to encrypt and select **Encrypt column data**.
   Result
   The **Encrypt Column** window is displayed.
3. Enter an encryption key in the **Encryption Key** field to encrypt the data in the column.

   Remember: When you have already encrypted other columns in the dataset, you must enter the same encryption key that you used previously. You can use only one encryption key to encrypt columns in a dataset.

   Important: The encryption keys you use to encrypt data in a dataset are not stored on the server nor can be retrieved from the server. Therefore, you must remember to store the encryption keys in a secure location. You must use the same encryption keys to view the encrypted values, to decrypt data, or enable the use of the encrypted dataset during test runs.

4. Click **Encrypt Column**.
   Result
   Asterisks are displayed instead of actual data for the encrypted column.

Results

The dataset column is encrypted.

Decrypting a dataset column

To view the content of an encrypted dataset, you can decrypt the dataset. Removing encryption from a dataset revokes the protection offered to the test data.

Before you begin

You must have created a dataset with at least one encrypted column. See Creating a dataset in a workspace on page 640 and Encrypting a dataset column on page 659.

1. Double-click the dataset in the Test Navigator.
   Result
   The dataset is displayed in a browser.
2. Right-click encrypted cells that display the contents with asterisks, and then select **Decrypt column data**.
   Result
   The **Decrypt Column** window is displayed.
3. Enter the encryption key that you used to encrypt the data in the column in the **Encryption Key** field.
4. Click **Decrypt Column**.
   Result
   Asterisks are replaced with the actual data in the decrypted column.

Results

The encryption is now removed from the selected column in the dataset. When you run a test that uses a dataset that contains decrypted data, the variable data is substituted for the data in the recorded test without prompting for the encryption key.

Using a digital certificate store with a dataset

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

1. Open a test for editing. On the **Common Options** page, click **Add Dataset**.
2. Create a dataset with two columns that contains a list of the certificates in the certificate store and a list of passphrases for the certificates. To learn more about adding datasets, see Creating a dataset in a workspace on page 640. You can use the supplied KeyTool program to generate a list of names of certificates in a certificate store.
3. Select **Fetch only once per user**.
4. Save the dataset.
5. On the **Security** page, under Digital Certificates, click **Add**.
6. Select a certificate from the certificate store that you created previously.
7. Type the passphrase for the selected certificate.
8. When prompted to dataset the digital certificate, click **Yes**.
9. In the **Select dataset column** wizard, choose the dataset that you added previously, and substitute the appropriate columns for the certificate name and passphrase.
10. Save the test, and then add the test to a schedule.

Results

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

Navigating between a dataset and a test

After you have created a dataset or imported a comma-separate values (CSV) file into a dataset, you can navigate between the dataset and associated tests in the test editor. You can enlarge the test and the dataset, list the datasets

that a test uses, navigate from a row in a dataset to the corresponding element in the test, see the data for a page or request, and add or remove dataset references.

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. Do one of the following actions:

Option	Description
Maximize the test window	Double-click the test tab (for example, ). Do not click the **x**, or you will close the test. To return to the default perspective, click **Window > Reset Perspective**.
View the datasets that a test uses	In the **Test Contents** area, click the first line of the test, which is the test name.
Navigate from a row in a dataset to its corresponding element	a. In the **Test Contents** area, click the test name, which displays the dataset. b. Expand the dataset to display the rows. c. Double-click the row.
View the data for a page or request	In the **Test Contents** area, click the page or request.
To add a reference to a dataset	In the **Test Element Details** area, drag your cursor over the candidate, right-click, and select **Substitute > Select Data Source**. The **Select Data Source** window opens. If you have not already added the dataset to the test, click **Dataset**, and then add the new dataset.
Remove a reference to a dataset	In the **Test Element Details** area, drag your cursor over the reference, right-click, and select **Remove Substitution**.

3. Save the test, if you have made any changes.

Correlating response and request data

For a test to run correctly, a request that is sent to a server might need to use a value that was returned by a previous request. By ensuring that this data is correlated accurately, you can produce better performance tests.

Data correlation overview

A request can include data that was returned in the response to a previous request. Associating data in this manner is called *data correlation*.

Interactions with an application are typically related to each other. For example, consider the following interactions with a web-based application, in which each request depends on information returned from a previous response:

1. A payroll clerk types the web address for an application, which sends a login prompt. When the clerk logs in, the web server returns a page that indicates that login has succeeded and a unique session ID to the web browser that the clerk is using.
2. The clerk clicks a link on the returned page, which requests that the web server open the page for searching the employee database. The web browser includes the session ID when sending the request. Based on the session ID, the web server knows that the request comes from someone who is already logged on, and so opens the search form for the employee database. The clerk then searches for a specific employee. The web server returns a photograph of that employee and the employee's unique ID.
3. The clerk clicks a link that requests the web server to return the payroll record for the employee. With this request, the web browser sends two IDs:
    - The session ID, so that the web server knows that the request comes from some who is logged on
    - The employee ID, so that the web server can locate and return the correct information

In this example, request 2 depends on request 1, and request 3 depends on requests 1 and 2.

If you record these interactions in a test, before running the test with multiple users, you would vary the test data. For example, you would replace the user name and password values, the employee name search values, or both, with values that datasets contain. When you run the test, each virtual user returns a different employee payroll record, based on the contents of the datasets.

In a generated test, where data in a request depends on data that is contained in the response to a previous request, the request data is substituted from the response data on which it depends. The term for this internal linking of response and request data is *data correlation*. When you run a test with multiple users and varied data, data correlation is required to ensure that the test runs correctly.

A *reference* is a value in a test (typically in a response) that can be used by a subsequent value in the test (typically in a request). When the test generator detects that a request value must be substituted from a previous value, it designates the earlier value as a reference and correlates the subsequent request value with the reference. This process is called *automated data correlation*. You can also manually correlate any two values in a test or unlink existing correlations.



Note: You can change or disable automated data correlation. To do so, click **Window > Preferences**, expand **Test**, and then click **Test Generation**.

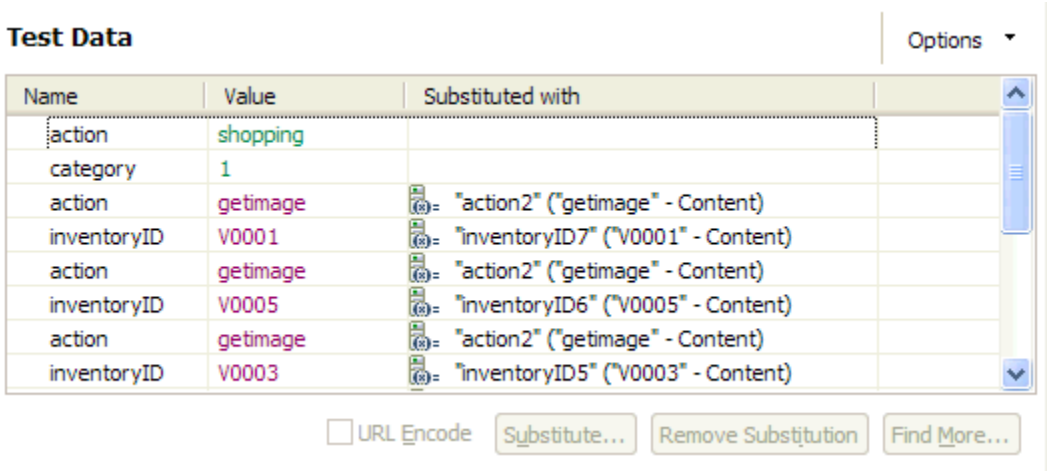
By default, the empty strings are not correlated because it might increase the time taken to generate a test. However, sometimes empty strings such as spouse name or middle initial become important to correlate. To correlate the empty strings, click **Window > Preferences > Test > Test Generation > HTTP Test Generation > Data Correlation** and select the **Create substitutions for empty strings** check box.

Generally, the HTML response content after the recording appears as *<input type="username" name="User" id="aaa" value="John"/>*. Some applications dynamically update the *name* attribute. Therefore, when you play back the test, the HTML response content appears as *<input type="username" name="idt020" id="aaa" value="John"/>*. Because the *name* attribute changes dynamically, data correlation does not occur and the playback fails. For data correlation to

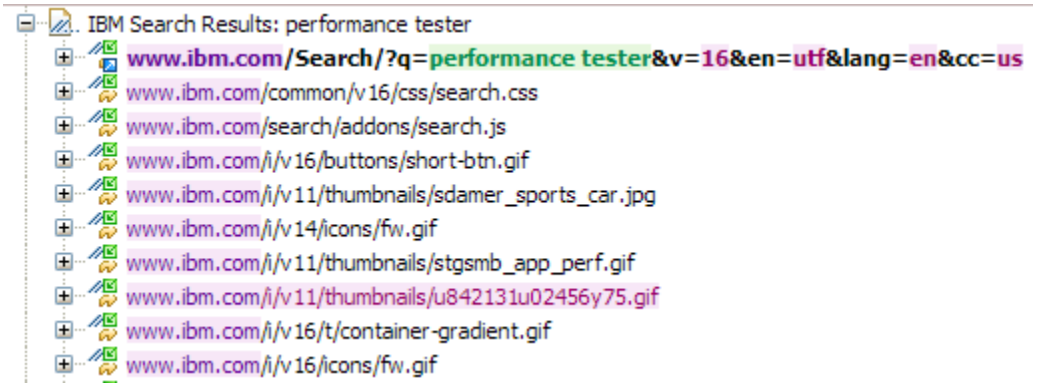
correlate the response content based on the *ID* attribute, ensure that you have selected **ON** in the **Priortize correlation based on ID** option at **Window > Preferences > Test > Test Generation > HTTP Test Generation > Data correlation**.

To help you work with correlated data, the test editor uses color coding and provides navigational aids:

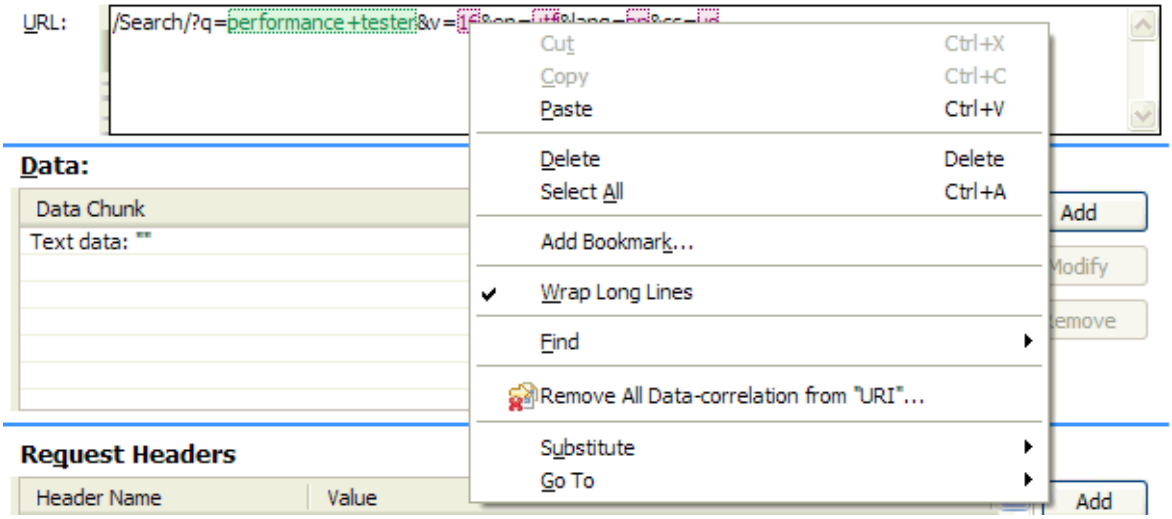
- When you click a page, you see a Test Data table for that page. By default, related dataset candidates are shown in green text on a light green background, values that are already associated with a dataset are shown in white text on a green background, and references are shown in blue text.



- If correlated data is not displayed, right-click the table and verify that **Show References** is selected. To navigate directly to a page request containing correlated data, double-click a table row. To associate correlated data from this table with a dataset, click the row, click **Substitute**, and then click **Select Data Source** to open the **Select Data Source** window. You can also use the **Test Data Sources** view to make substitutions. In the test editor, right-click the **Test Data** table, and then select **Link with Test Data Sources View**. When you click a row in the **Test Data** table, the **Test Data Sources** view displays information about the selected substitution site.
- When you expand a page, green text indicates page requests that contain dataset data or candidates. Blue text indicates page requests that contain references.



- When you click a highlighted request, dataset candidates are highlighted in light green, data that is associated with a dataset is highlighted in dark green, and correlated data is highlighted in red. If you right-click a value for correlated data, as shown in the example, you can then click **Go To** to see its reference:



- References are highlighted in dark blue.

Viewing data correlation

You can switch between viewing all test elements in the test editor and viewing only elements related to data correlation in the test editor. Viewing only data correlation elements makes it easier to add and remove substitutions.

1. In the Test Navigator, browse to the test, and double-click it. The test opens.
2. In the **Test Contents** area, click **Options**.
3. Click **Show > Data Correlation**.
   Result
   The test editor window displays only elements that are related to data correlation. Alternately, click **View** under **Test Contents** to switch between **Display all Test Contents** and **Show Substitutions**.
4. Select a single test element in the **Test Contents** area to see the current data source and to remove or change the substitution in the **Test Element Details** area. Select multiple elements in the **Test Contents** area to see the data in tabular form in the **Test Element Details** area. Different controls are available depending on the type and number of elements that you select in the **Test Contents** area.
5. **Optional:** In the Test Elements Details area, click **Substitute > Select Data Source** to open the **Select Data Source** window, where you can specify the data source for the selected substitution site.

What to do next

To view all test elements, click **Options > Show > Data Correlation** again.

Note: If you select a test element while viewing all test contents, and then switch to viewing only data correlation elements, then the corresponding substituters and dataset candidates are selected. For example,

if you select an HTTP page in the test editor, and then switch to viewing only data correlation elements, then all substituters and dataset candidates for all requests from the HTTP page are selected.

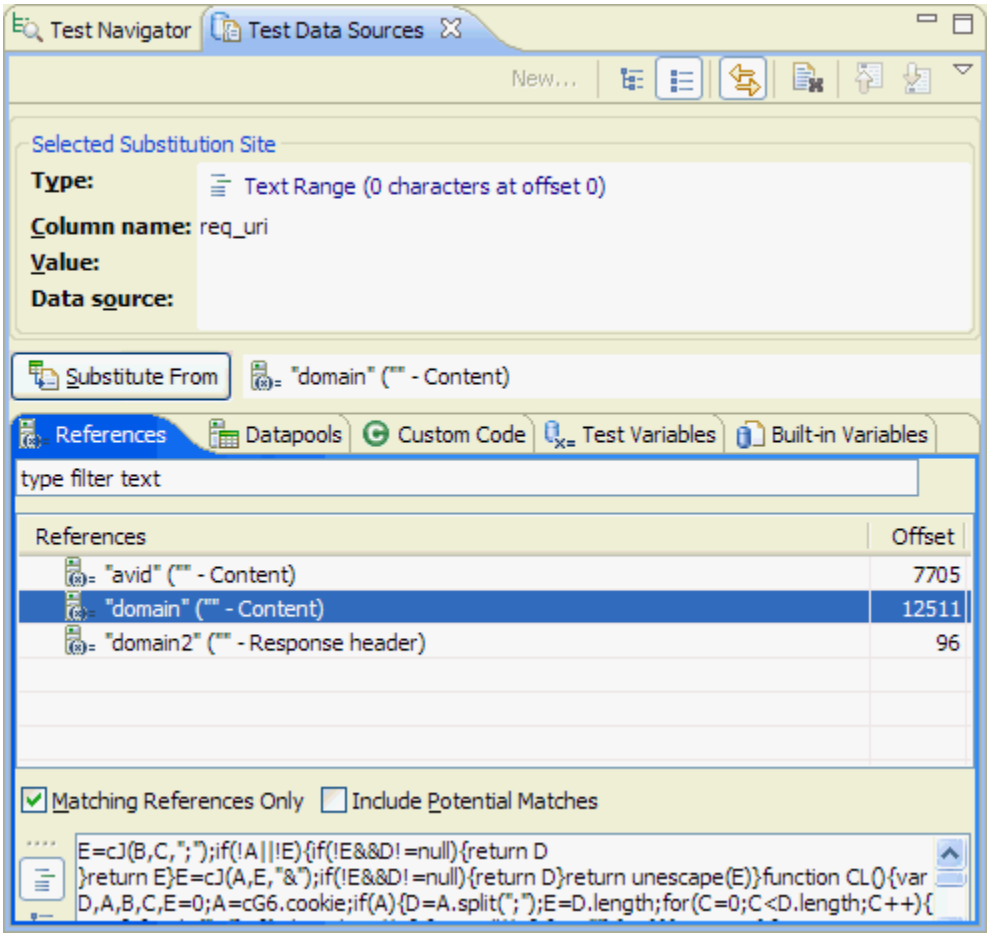
Test data sources overview

Use the **Test Data Sources** window to add or change data correlation for any supported test element.

The test generator attempts to perform automated data correlation. You can also manually correlate request values with other test data. The **Test Data Sources** window displays the following types of test data that you can substitute from:

- References
- Datasets
- Custom code
- Test variables
- Built-in datasources

You can right-click references, custom code, and built-in data sources to display a menu that contains commonly used commands. In addition, you can use the toolbar and menu at the top of the **Test Data Sources** window to complete common operations, such as creating a dataset or switching from tree view to list view. The **Substitute From** push button is enabled when you select a substitution site in the test editor and a data source from one of the five categories in the **Test Data Sources** window. Click **Substitute From** to correlate the data source and the substitution site.



References

The data sources that have been created in the test and the potential matches that are available for a selected substitution site. For example, text that is found in a response and used in a subsequent request is automatically created as a reference. Clear the **Matching References Only** check box to see all the references that occur before the substitution site in the test. Select **Include Potential Matches** to see a list of locations that might match the substitution site. Replace the `type filter text` string with keywords to filter the list of references. Select a reference in the **References** table to display the reference in the pane at the bottom of the window. Use the controls to the left of the preview pane to switch between inline view and tree view.

Datasets

The datasets that have been added to the test. To add a new dataset, click the **Add new Dataset** icon ( ).

Custom Code

The custom code that is available as data sources for this substitution site, if you have written Test Execution Services (TES) custom code.

Test Variables

The test variables and their types that are available as data sources for this substitution site. Replace the `type filter text` string with keywords to filter the list of test variables.

Built-in Datasources

The dynamically calculated data sources, such as **Current Date**, **Random Number**, **Sequential Number**and **Timestamp**, that are available as data sources for this substitution site. To create a built-in data source, right-click inside the **Built-in Datasources** page, and select **New**, or click the **New** push button at the top of the window.

To assign a unique value for every substitution, when creating the built-in data sources, select the **Get New Value Each Time Used** check box.

Detaching the Test Data Sources window

The **Test Data Sources** window is detachable. To detach the **Test Data Sources** window, right-click the **Test Data Sources** tab, and then select **Detached**. When detached, the **Test Data Sources** window is always displayed in front of the other windows that make up the workbench.

Correlating a request value with test data

If a test runs without error but does not generate the results that you expect, you might need to correlate a value in a request with other test data.

About this task

You can correlate a request value with the following types of test data:

- References
- Datasets
- Custom code
- Test variables
- Built-in datasources

For example, if you recorded a test and searched on a date, you might want to substitute the built-in data source *Current Date* so that the test will search on the playback date, not on the recorded date. For information on the different types of test data, see Test data sources overview on page 665.

1. In the **Test Navigator**, browse to the test, and double-click it. The test opens in the test editor.
2. Locate the value that the other test data will replace.
3. Highlight the value: Press and hold the left mouse button and drag your mouse over the value.
4. In the **Test Data Sources** window, click the appropriate tab: **References**, **Datasets**, **Custom Code**, **Test Variables**, or **Built-in Datasources**. To see all references, clear the **Matching Only** check box.
5. In the **Test Data Sources** window, select the test data to use. For references and custom code, you can double-click the entry in the **Test Data Sources** window to find the data source in the test editor.
6. Click **Substitute From**.

Result

The value is shown in purple text to indicate that it has been correlated and the correlation is added to the Test Data table, which contains the substitution sites for the page.

Substituting request content with file contents

You can substitute the content portion of a protocol request with the contents of a file. This feature is only supported in certain sections of a test, depending on the protocol. For example, HTTP tests support file content substitution only in the POST data section of a request. SOA tests support file content substitution for MIME and DIME attachments, XML node values and fragments, and text content. File content substitution works in the same manner as other substitutions. All standard test data sources such as test variables, datasets, and references can be used. The data from the source is treated as a full path to a file. The file is opened, its contents are read, and then those contents are used in the substitution.

Before you begin

You must copy the files that contain the substitution content to the agent computers. You must record a test with locations for substitution from a file. For example, record an HTTP test that contains multipart MIME data in a POST request.

1. Create a data source that contains the full path to the file from which you want to substitute content. Specify an absolute path to the file. Use path separator characters appropriate to the operating system of the agent computer running the test. Optionally, specify a character set to use in reading the file. The existence of the file is not validated. If the file cannot be opened when the test runs, a message is written to the test log. If you use path separator characters that are not appropriate for the operating system of the agent computer, the substitution cannot be completed. For example, if you use a path of `D:\DataFiles\file1` on an agent computer running Linux™, the substitution cannot be completed, because Linux™ uses forward slashes as path separator characters.
2. In the test editor, navigate to the request where you want to substitute content, and then select the request data that you want to substitute.
3. Right-click, and then select **Substitute > Select Data Source**.
4. Select the data source that contains the path to the file from which you want to substitute.
5. Right-click the substitution site, and then select **File Contents Substituter**.

Results

When the test runs, the content in the protocol request is substituted with the specified file contents.

HTTP POST data is displayed in the test editor in chunks. You can create a file contents substitution in the POST data of an HTTP POST request by selecting the data chunk that you want to correlate, and then clicking **Substitute**. The test data source that you select is automatically treated as a file contents substituter. The entire data chunk is replaced with the contents of the file when the test runs, even if only a portion of the text in a text data chunk is selected by the substituter.

Built-in Datasources

You can use built-in data sources instead of creating custom data sources to substitute the recorded values. The Built-in Datasources in the **Test Details** section displays the data sources that have been used and unused. You can also modify their properties from a location.

The built-in data sources are Current Date, Random Number, Sequential Number, and Timestamp. The values of these data sources are dynamically calculated and submitted to the test.

Assigning random numbers to users or clients

To assign unique random numbers to all the virtual users in a test, you can create a random number data source in the workbench. This data source will generate unique integers or floating point numbers for the users or clients. You can choose to distribute the numbers of the virtual users or clients in a uniform, normal, or negative exponential way.

1. In the Test editor, select the root node (name of the test) and from the Test Details area, select **Built-in Datasources**.
2. Select the **Random Number** data source and click **Add Built-in Datasource**.
3. Type a name for the data source.
4. Select how do you want to generate the numbers:
    - **Uniform**: Click this option to generate random numbers with a uniform distribution. Specify the minimum and maximum values for the generated numbers.
    - **Normal**: Click this option to generate random numbers with a normal or Gaussian distribution. Specify the average and the standard deviation for the generated numbers.
    - **Negative Exponential**: Click this option to generate random numbers with an exponential distribution. Specify the average for the generated numbers.
5. Select how do you want to format the numbers. You can select **Common** to format the numbers in decimal or scientific notations.
   You can choose **Custom** to specify a **Format mask** by using the standard Java formatting syntax. The changes that you specify can be previewed in **Formatted output**.
6. To substitute the built-in data source every time with a new value for the requests, select the **Get new value each time used** check box.

Assigning sequential numbers to users

To assign unique sequential numbers to all the virtual users in a test, you can create a sequential number data source in the workbench. This data source will generate unique integers or floating point numbers for the users.

Before you begin

You can use the sequential built-in data source option wherever data correlation substitutions are permitted, such as Transactions or Delays.

About this task

You define an initial value that should be assigned to the first virtual user and a step value that is a number by which the current value increments after each retrieval by a virtual user. If initial value is 1 and step value is 5, the workbench

generates numbers in the sequence of 1, 5, 10, 15, and so on and each number is mapped to a virtual user. If a test is run on multiple agent machines, the workbench assigns a sequential value to all the users in all the agent machines.

You can also assign a full sequence of numbers of one virtual user. The sequence number increments in the request for each time the request in the multi-request generator is executed.

1. In the Test Contents area of the test, click an element in the test where data correlation substitution is permitted, such as a transaction name or delays.
2. In the Test Element Details area, right-click the name of the element and click **Substitue > Built-in Datasources**.
3. In **Built-in Datasource Selection Wizard**, click **Sequential Number** and click **Next**.
4. Assign a name for the data source.
5. In **Initial Value**, type a number to be assigned to the first virtual user.
6. In **Step Value**, type a number.
7. In **Formatting Options**, you can format the number in the manner you want to use.
8. **Optional:** To assign a full sequence of numbers to one virtual user, select the **Execute for individual user** check box.
9. **Optional:** To substitute the built-in data source every time with a new value for the requests, select the **Get new value each time used** check box.
10. Click **Finish**.
11. Save and run the test.

Results

After you add the test to a schedule and run the schedule, the test log displays each element name where the data source is applied with the unique sequential number assigned to the virtual user.

Correlating multiple fields in a test

Some tests are structured in such a way that you must correlate data for multiple fields.For example, assume that you plan to dataset an item that a virtual user is buying. For the test flow to be correct, you must also dataset all occurrences of that item in the test. You can find and correlate all instances of that item in one procedure. Typically, you use **Find More and Substitute** in the **Show Dataset Candidates** window to correlate data for multiple fields. See Viewing dataset candidates when you open a test on page 652. Alternatively, you can use the **Test Search** page to correlate data for multiple fields.

To find all instances of a field in a test and correlate some or all of the instances with a data source, such as a dataset:

1. In the Test Navigator, browse to the test, and double-click the test. The test opens.
2. Locate the item or the substitution site to change or create a reference for. If the item is plain text, select the item. If the item is an existing reference, click the highlighted area.
3. Right-click, and then click **Find > More Substitution Sites**.
4. Click **OK**.

5. On the **Test Search** page, select **Case sensitive** to perform a case-sensitive search or **Regular expression** to perform a search using regular expressions. In regular expression mode, press Ctrl+spacebar key in **Search for text** for content assistance. Content assistance lists the regular expression patterns and the content that they match.
6. Click **More Options**, and then select the appropriate options:

   Restrict to elements highlighted in Test Contents

   Search only in elements that are selected in the **Test Contents** area.

   Highlight found elements in Test Contents

   Highlight found elements in the **Test Contents** area.

   Recursive

   Searches the child test elements in addition to the element. For example, if you search an HTTP page, select this option to search the requests and responses within the page.

   Match encoded and decoded values (protocol-specific)

   When selected, searches for matches of the unencoded and URL-encoded versions of the specified text. For example, when searching in HTTP data, `abc%123` and `abc%25123` match.

   Include matches with overlapping data correlation

   Include sites that are contained in, or overlap with, an existing substitution site. If you decide to substitute, the conflicting substitutions are automatically removed.

   Include matching substituters

   Click to return elements that originally matched the search string but have since been substituted. Clear to skip existing substitution sites when results are returned.
7. Click **Close**.
8. Click **Search**. The search results are displayed in the **Search** view.
9. In the **Search** view, select the matches to substitute, and then right-click the selection.
10. Optional: To select all matches, right-click the test name.
11. Click **Substitute in DataSource View**.

    Result

    This action sends the selected matches to the **Test Data Sources** window.
12. In the **Test Data Sources** window, click the tab that corresponds to the type of data source to use:

Option	Description
References	The data sources that have already been created in the test and the possible matches that are available as data sources for the selected substitution site. For example, text that is found in a response and used in a subsequent request is automatically created as a reference. Clear the **Matching References Only** check

Option	Description
	box to see all the references that occur before the substitution site in the test.
Datasets	The datasets that have been added to the test. To add a new dataset, click the **Add new Dataset** icon ().
Custom Code	If you have written test execution services (TES) custom code, the custom code that is available as data sources for this substitution site.
Test Variables	The test variables and their types that are available as data sources for this substitution site.
Built-in Datasources	The dynamically calculated data sources (**Current Date**, **Random Number**, **Sequential Number**, and **Timestamp**) that are available for this substitution site. To create a new built-in data source, right-click inside the **Built-in Datasources** page, and select **New**.

13. Select the data source, and click **Substitute From**.
    Result
    The **Substitute Multiple Items** window is displayed, showing information about the data source and substitutions sites that you selected.
14. For each site with a selected check box, click **Substitute Checked** to substitute the data source or clear the check box to skip the site.
    Click **Always Prompt** to examine every substitution site one at a time. Click **Prompt on overlapping data correlations** to examine a site only if the site you are substituting into is contained in, or overlaps with, an another substitution site. If you decide to substitute, the conflicting substitutions are automatically removed.

Results
The selected instances of the field are correlated with the data from the data source.

Guidelines for adjusting 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 can cause these problems.

To identify data correlation problems:

1. Use the **Potential Correlation Errors** view to find missing or incorrect data correlations. See Finding data correlation errors on page 675 for more information.
2. Run a test individually or in a schedule with the **Log Level** for errors, failures, and warnings set to **All**.
3. After the run, open the test log as explained in Viewing the test logs on page 1493.

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

- **Insufficient correlation:** Test values that must be correlated are not. Some possible causes follow:
  - Two parameters that must be correlated have different names.
  - A value must be correlated with a previous value that does not occur in the expected location.
  - A parameter or value must 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 are correlated.
- **Incorrect correlation:** Test values that must be correlated are correlated incorrectly.

Insufficient correlation: Parameters have different names or occur in unexpected locations

When two parameters that must be correlated have different names, automated data correlation does not recognize that the two parameters are related. For example, consider this request: http://www.example.com?id=12345. Suppose that this request must be correlated with the server response that contains `customer_ID=12345`, not `ID=12345`. In this case, the `ID` parameter must be correlated with `customer_ID`.

Data correlation typically links a response value that was returned from the server with a subsequent request value. The automated correlation algorithms search in the URL and the POST data for potential matches; however, other schemes for returning parameters are possible. For example, consider this request: http://www.example.com?id=12345. Suppose that this request must be correlated with the server response that contains the name and entity pair `href name="customer_ID" entity="12345"`, not `ID=12345`. In this case, the `ID` parameter must be correlated with `name="customer_ID"` and value `12345` must be correlated with `entity="12345"`.

Here are some additional causes of insufficient correlation:

- SOAP designates correlation parameters in external XML files. The correlation algorithms cannot correlate parameters in the external file with parameters in the test.

To manually correlate data in these cases:

1. In the test editor, use search or browse to locate the two parameters for correlation.
2. Navigate to the parameter that occurs later in the test, and select the parameter. This is the substitution site.
3. In the **Test Data Sources** window, click the **References** tab.
4. Select the data source to use as a reference, and then click **Substitute From**.

Insufficient correlation: One parameter is unnamed

Sometimes a parameter or value must 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 must understand how and where the parameter or value is computed, and then use a custom code block. See Extending test execution with custom code on page 629 for more information about custom code.

For example, consider the web address http://www.example.com?login_stamp=12345_Apr_11_07, where the value for `login_timestamp` is the concatenation of the login ID and the current date. In this case, you must generate a 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="Apr_11_07"`. In this case, you can put these parameters in separate references and, in subsequent requests that use 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:

1. In the test editor, search or browse to locate the substitution site that must not be correlated. By default, purple letters indicate correlated data.
2. Right-click the substitution site.
3. Click **Remove Substitution**.

Incorrect correlation

A parameter that requires 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 use the wrong instance.

With the HTTP Test Generation preferences, you can optimize automatic data correlation for accuracy or for efficiency.

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

Note: If you do not manually apply a correlation in the Referer field in an HTTP request header, then the Referer field is automatically correlated as needed. If you manually apply a correlation in the Referer field in an HTTP request header, then no automatic correlation is performed.

Incorrect correlations are more likely to happen when **Optimize automatic data correlation for execution** is set to **Efficiency**. To fix an incorrect correlation:

1. In the test editor, search or browse to locate the value that is incorrectly correlated.
2. Right-click the substitution site.
3. Click **Remove Substitution**.

4. Right-click the substitution site again.
5. Click **Substitute**, and select the correct parameter.

Generally, the HTML response content after the recording would look like `<input type="username" name="User" id="aaa" value="John"/>`. Some applications dynamically update the name attribute. So, when you play back the test the HTML response content would look like `<input type="username" name="idt020" id="aaa" value="John"/>`. Because the name attribute is changing dynamically, data correlation would not occur and the playback would fail.

Such correlations are the result of the tool using the *name* attribute as the basis for correlating other attributes in the response code instead of the *ID*. To correlate the responses based on ID, select **ON** in **Window > Preferences > Test > Test Generation > HTTP Test Generation > Data correlation types > Prioritize correlation based on ID**.

Finding data correlation errors

You can use the **Potential Correlation Errors** view to find missing or incorrect data correlations.

Before you begin

Run a test or a single-user schedule. The **Potential Correlation Errors** view does not support multiple-user schedules. If verification points fail while you are running a test, you are prompted to open the **Potential Correlation Errors** view when the test run is complete.

To find data correlation errors:

1. In the **Test Navigator**, select the result of the test run where you want to find correlation errors.
2. Right-click the result, and then select **Find Data Correlation Errors**. You can choose **Missing Correlation**, **Incorrect Correlation**, or **All**.
3. The **Potential Correlation Errors** view opens.
   After the test log is processed, the view is populated. Depending on the size of the test log, it can take significant time to populate the view. The potential missing or incorrect data correlations are displayed, in descending order of the likelihood that the correlation is incorrect. Selecting an item in the **Potential Correlation Errors** view automatically selects the corresponding element in the test editor, so that you can fix the potential error.
4. Use the **Compare with Test Log** toolbar button in the upper-right corner of the view to compare the request or response in the test with the same object in the test log.
5. For missing correlations, use the **Suggest Fix** toolbar button in the upper-right corner of the view to search for other instances of the value in all responses in the test. If a matching value occurs in an earlier response in the test, create a reference in that response.

Disabling data correlation

You can disable a data correlation source or a substitution site. When you disable a data source, none of the substitution sites that use the source will be correlated when you run tests. When you disable a substitution site, only that specific substitution site is disabled. Other substitution sites that use the same reference will be correlated when you run tests. You can also disable data correlation entirely for subsequent tests that you record.

To disable a data correlation source or substitution site:

1. In the **Test Navigator**, browse to the test and double-click it. The test displays in the test editor.
2. In the **Test Contents** area, click a request.
3. In the **Test Element Details** area, locate the data correlation source or substitution site.
4. Right-click the data value and select **Disable** from the menu.
   To re-enable a disabled data source or substitution site, right-click the data value and select **Enable** from the menu.

Results

The data correlation source or substitution site is disabled.

Note: To disable data correlation for the entire workspace, click **Window > Preferences > Test Generation**, and clear **Enable automatic data correlation**. Subsequent tests that you record or regenerate will not include data correlation.

Recorrelating test data

If you disabled automatic data correlation before recording a test, you can regenerate the test with automatic data correlation enabled.

1. Click **Window > Preferences > Test > Test Generation**.
   Result
   The **Test Generation** preferences window opens.
2. Click the **Data Correlation** tab.
3. Select the types of data correlation to enable, and then click **OK**.
4. In the **Test Navigator**, browse to the test and double-click it. The test displays in the test editor.
5. Click **Edit > Re-correlate test data**.

Results

The test is regenerated with the types of automatic data correlation that you selected.

Specifying error-handling behavior

You can specify how error conditions are handled when running a test or schedule. Error conditions include verification point failures, connection failures, server timeouts, custom code alerts, and problems with data correlation.

About this task

You can specify error-handling behavior for the workbench, schedules, tests, and test elements/steps. The Errors report displays the error conditions and error behavior that occurred in a test or schedule. When you set the error-handling behavior for the workbench, it is applied to all the tests in the workbench. To apply it for the workbench, click **Window > Preferences > Test > Test Execution > Error handling**.

To stop running a test when a substitution failure occurs in data correlation, select the **Substitution Failure** check box. Click **Exit Test**, and then click **OK**. The error-handling behavior that is specified closest to where an error occurs takes precedence. If a specific request is set to continue if a substitution fails, and the schedule is set to stop if a substitution fails, then the schedule will continue running if the substitution fails in that request.

For example, to stop running a test when a Mobile or Web UI action fails, select the **UI Action Failed** check box. Click **Exit Test**, and then click **OK**. The error-handling behavior that is specified closest to where an error occurs takes precedence. If a specific action is set to continue when a step fails, and the test is set to stop if a step fails, then the test will continue running if the step fails.

1. Double-click the test to open it from the **Test Navigator** view.
   Result
   The test opens.
2. In the Test editor, complete one of the following steps:
   a. To specify error-handling behavior for a request/step or other elements in the test hierarchy, in the **Test Contents** section, select the element and in the **Test Details** section, click the **Advanced** tab.
   b. To specify error-handling behavior for an element such as a connection, verification point, substitution, or reference, select the element, and then click **Change**. Skip to step 5 on page 679.
   c. To specify error-handling behavior for a dataset that is associated with a test, on the **Common Options** page, select the dataset and click **Dataset Properties**, and then click **Change**. Skip to step 5 on page 679.
3. Under **Error Handling**, expand **Click to show conditions**.
   Result
   The error condition table is displayed. The error condition table shows all possible conditions that can be handled in the test.

Error condition	Description
Page Title Verification Point Failure [HTTP]	The returned title for the primary request for an HTTP page does not match the expected title. The default value of the expected page title is what is returned between the <title></title> tags during recording. See Specifying the expected page title on page 680.
Response Code Verification Failure [HTTP]	The returned response code does not match the expected response code. You can specify an exact match or a relaxed match. See Specifying the expected response code on page 681.
Response Size Verification Failure [HTTP]	The number of bytes returned does not match the expected number of bytes. You can control how closely the returned response size must match the recorded response size. See Specifying the expected response size on page 682.
Content Verification Point Failure	The received data does not match the expected data. Content verification point controls are protocol-specific.

Error condition	Description
Connection Failure	The workbench or agent computers cannot connect to the server under test.
Authentication Failure	An attempt to log in to the server under test failed.
End of Dataset reached	The last row of the dataset is reached. See Dataset overview on page 636 for more information.
Test Verdict Failed	The error condition Test verdict failed is met if the test verdict is set to fail.
Reference Extraction Failure	The response received during playback is different from the response received when the test was recorded. Data correlation failed because the regular expression that is associated with the reference did not match the expected value.
Substitution Failure	A reference for an expected data substitution is a null reference.
Server Timeout	The server under test does not respond before the timeout interval elapses.
Dropped Connection	When the system under test drops the connection.
Custom Verification Point Failure	A custom verification point did not return a Pass status after performing a verification written in Java™ code. See Reporting custom verification point failures on page 1068.
Custom Code Alert	Custom code reported an **Alert** condition. The following code reports a custom code alert: `tes.getTestLogManager().reportErrorCondition(RPTCondition.CustomCodeAlert);` See the ITestLogManager Javadoc for more information. The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.
Custom Code Exception	The custom code in a test has an exception. Rational® Functional Tester-HCL OneTest™ UI logs you out whenever there is an exception in the custom code. For information on setting different actions. See Error-handling behavior on page 676.
UI Action Failed	This option is to stop running a test when a Mobile or Web UI action fails.
UI Fatal Error	This option is to allow test execution for a compound test to continue even after a fatal exception in one of the tests in a compound test.

4. Select the check box next to the condition for which to specify behavior.

   Result

   A window opens where you can specify the action to take and the message to log when the condition occurs.

5. To apply an action when the specified condition occurs, select the **Override action upon error** check box .

Option	Description
Continue	Click to continue running the test.
Exit transaction	Click to exit a transaction when the specified condition occurs. Select the **Innermost** or **Outermost** transaction, or type the name of a transaction.
Exit loop	Click to exit a loop when the specified condition occurs. Select the **Innermost** or **Outermost** loop, or type the name of a loop.
Continue to next iteration of loop	Click to continue to the next iteration of a loop when the specified condition occurs. Select the **Innermost** or **Outermost** loop, or type the name of a loop.
Exit test	Click to exit the test when the specified condition occurs.
Exit user	Click to stop the virtual user that encounters the specified condition. For a Web UI test, this action will stop the test execution.
Terminate run	Click to stop the run when the specified condition occurs.

6. To contribute to the health of the page, transaction, or loop, select the **Override contribution to health status** check box and select **Yes**. The respective reports display the health of the page, transaction, or loop.
7. **Optional:** To write a message to the test log when the specified condition occurs, select the **Override log message upon error** check box and type a message.
8. Click **OK**.

Inserting a custom log message into a UI test script

You can insert a custom log message into the test script of Web UI, Windows, or mobile tests to differentiate a set of steps related to a task in the unified report. For example, you can add custom messages to demarcate the test steps related to order generation, user creation, or report generation in an application.

Before you begin

You must have completed the following tasks:

- Recorded a Web UI, Windows, or mobile test. See Testing in the UI Test perspective on page 532.
- Read and been familiar with the usage of variables. See Declaring and assigning test variables.
- Read and been familiar with the substitution of values by using a dataset. See Associating a dataset with the test.

1. Expand the **Tests** folder to locate the required test in the **Test Navigator** pane.
2. Double-click the test to open it.
3. Select the test step for which you want to insert the custom log message, and then click **Insert > Log Message**.

Or right-click the step, and then click **Insert > Log Message**.
A Log Message step is inserted before the selected test step.

4. Enter the text that you want to include in the report in the **Message** text box under the **Log Message Details** pane.
5. **Optional:** Select the text in the custom log message and substitute it with the value of a variable or a dataset. During the playback, the value of the variable or dataset replaces the text marked for substitution. You can view these substituted custom log messages in the unified report after the test run.
6. Click **File > Save** to save the changes.

Results

You have inserted a custom log message into the test script.

What to do next

You can run the test and verify the inserted log message in the unified report. See Running a Web UI test on page 1243.

Verifying expected behavior

To check whether an expected behavior occurred during a run, you add verification points. When you run a test that contains a verification point, an error is reported if the expected behavior did not occur. When global verification points are disabled (the default), you can enable verification points for a specific test.

Enable verification points globally. Click **Window > Preferences > Test > Test Generation > HTTP Test Generation**.

Specifying the expected page title

Page title verification points verify that the primary request for a page returns the expected page title. If the returned title is unexpected, the test log reports a failed verdict event. Although the comparison is case-sensitive, it ignores multiple white-space characters (such as spaces, tabs, and carriage returns).

1. In the Test Navigator, browse to the test and double-click it.
   Result
   The test opens.
2. 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.
3. Click the page title to display the editing fields in the **Test Element Details** area.
4. 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.

What to do next

You can also change the preferences so that Page Title verification points are set automatically. To do this:

1. Change the HTML page title. Click **Window > Preferences > Test > Test Generation > HTTP Test Generation > Verification Points**, and select **HTML Page Title**. This changes subsequent tests that you record.
2. Optionally, regenerate existing tests with the changed preference, as shown in Generating a new test from a recorded session.

Specifying the expected response code

Response code verification points verify that the response code matches an expected value. If the returned code is does not match, the test log reports a failed verdict event. You can specify an exact response code or verify that the code is within the same category.

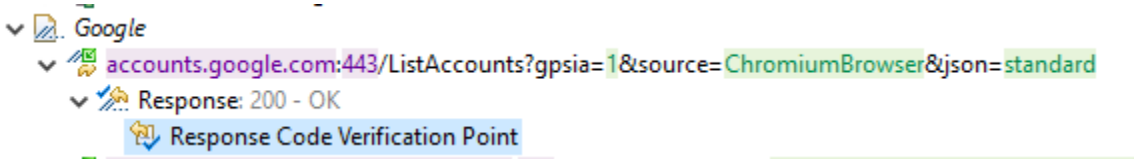
About this task

You can either change the preferences so that Response Code verification points are set automatically for all the tests or you can configure each test setting. To set response code in the preferences:

1. Change subsequent tests that you record. Click **Window > Preferences > Test > Test Generation > HTTP Test Generation > Automatically include verification point of**, select **HTTP Return Code** and click **Relaxed** or **Exact**. This changes subsequent tests that you record.
2. Optionally, regenerate existing tests with the changed preference, as shown in Generating a new test from a recorded session.

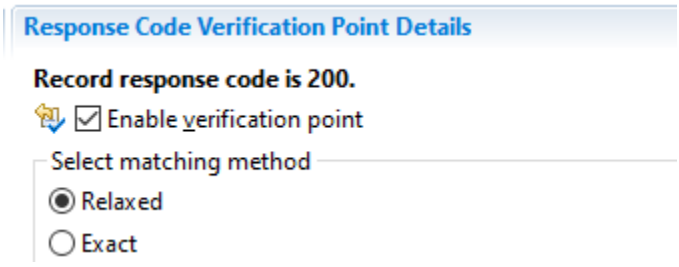
To set response code setting for a test:

1. In the Test Navigator, browse to the test and double-click it.
   Result
   The test opens.
2. 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. The following figure shows a response code verification point within a test:



Note: When you modify a test, the modified items appear in italic type. The italic type changes to regular type after you save the test.

3. Click the verification point to display the response code editing fields in the **Test Element Details** area.



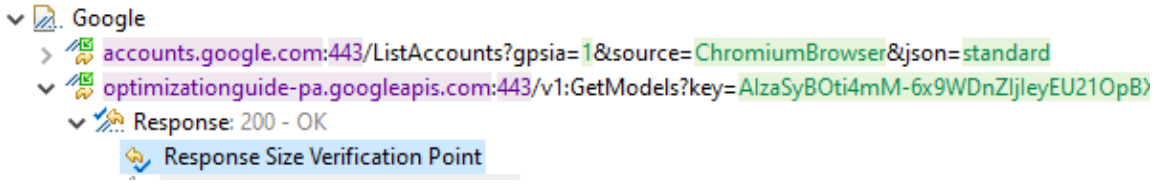
4. To disable an individual response code verification point, clear the **Enable verification point** field.
5. From the **Select matching method** list, click an option to indicate how closely the returned response code must match the recorded value.

Option	Description
Relaxed	If the recorded response code is 200, 201, 202, 204, 206, 301, 302, or 304, then a returned response code of any of those values causes the verification point to pass. If the recorded response code is any other value, it is the same as if you had specified an exact match.
Exact	An error is reported if the returned response code does not match the recorded value.

Specifying the expected response size

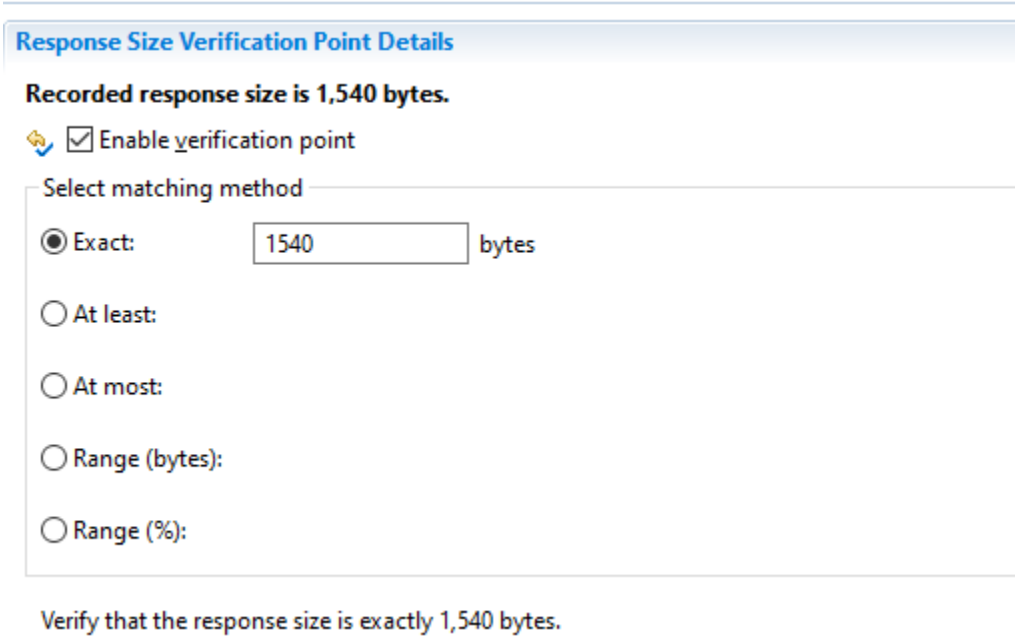
Response size verification points verify that the number of bytes returned in a response is what you expected. If the byte count does not match, the test log reports a failed verdict event. You can specify an exact response size or verify that the byte count is within a range.

1. In the Test Navigator, browse to the test and double-click it.
   Result
   The test opens.
2. 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. The following figure shows a response size verification point within a test:



Note: When you modify a test, the modified items appear in italic type. The italic type changes to regular type after you save the test.

3. Click the verification point to display the response size editing fields in the **Test Element Details** area.



   Responses to HEAD requests, by definition, have no contents; the size is always 0.

4. To disable an individual response size verification point, clear the **Enable verification point** field.
5. From the **Select matching method** list, click 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.

What to do next

You can also change the preferences so that Response Size verification points are set automatically. To do this:

1. Change subsequent tests that you record. Click **Window > Preferences > Test > Test Generation > HTTP Test Generation > Verification Points**, and select **HTTP Response Size**. This changes subsequent tests that you record.
2. Optionally, regenerate existing tests with the changed preference, as shown in Generating a new test from a recorded session.

Frequently asked questions

This section provides answers to frequently asked questions about the IBM® Rational® Functional TesterHCL OneTest™ UIWeb UI extension.

FAQ: Web UI testing

This topic answers a few generic questions about using the IBM® Rational® Functional TesterHCL OneTest™ UI Web UI extension.

- Why do I see the phrase WebDriver Text displayed in the bottom right corner of the Mozilla Firefox browser while playing back a Web UI test script? on page 684
- Why am I unable to record a Web UI test using Microsoft Internet Explorer? on page 684
- Why does it take so long to start recording a Web UI test using Firefox? on page 684
- What can I do if the browser does not start during recording? on page 684
- Why do I see variations in the recording and playback of a Web UI test on different browsers? on page 685
- Why does the recorder stop abruptly while recording in Microsoft Internet Explorer on a Windows computer? on page 685

Why do I see the phrase `WebDriver Text` displayed in the bottom right corner of the Mozilla Firefox browser while playing back a Web UI test script?

The message originates from the Selenium API that the Web UI extension uses to open the Firefox browser. The Web UI application does not have any control over the display of the message, but it is not in any way impact the script playback. You can ignore the message.

Why am I unable to record a Web UI test using Microsoft Internet Explorer?

In Internet Explorer, open **Tools > Internet Options > Security** and disable protected mode for all zones.

Why does it take so long to start recording a Web UI test using Firefox?

The delay occurs because Rational® Functional TesterHCL OneTest™ UI must first invoke Selenium to start the Firefox browser.

What can I do if the browser does not start during recording?

1. Check that the internet proxy settings are set correctly. (In Internet Explorer, go to **Internet Options > Connections > LAN Settings** to make any necessary changes.
2. Close Rational® Functional TesterHCL OneTest™ UI.
3. Kill the following browser-specific processes:
    - InternetExplorerDriver.exe
    - ChromeDriver.exe
    - All FireFox instances

Why do I see variations in the recording and playback of a Web UI test on different browsers?

Recording and playback of a Web UI test depends on the properties of each control on a Web page. For example, in one browser a search button might be implemented as a text box, whereas in another browser the search button might be implemented as a jQuery control. To play back a recorded script in a different browser, you might need to edit the control recognition properties in the recorded script or insert a suitable action.

Why does the recorder stop abruptly while recording in Microsoft Internet Explorer on a Windows computer?

This is a limitation that the Web UI extension inherits from Selenium in case of non-admin accounts. When you are logged in as a non-administrator on a Windows computer, ensure that the Internet Explorer runs in unprotected mode. In Internet Explorer, open **Tools > Internet Options > Security**. Select **Internet** zone and clear the check box **Enable Protected Mode**.

Troubleshooting Web UI tests

While working with Web UI tests, you might encounter problems that you can easily troubleshoot. The topics in this section address some of these problems.

In addition to the topics listed in this section, the following resources are available to help you troubleshoot the Web UI problems:

- Support Knowledge Base: Known problems of Rational® Functional TesterHCL OneTest™ UI are documented in the form of individual tech notes in the Support Knowledge Base. As problems are discovered and resolved, the knowledge base is updated and maintained with new information. By searching the knowledge base, you can quickly find workarounds or solutions to problems.
- Frequently Asked Questions on page 684

You can contact IBM Software Support if you are unable to troubleshoot the problem. Gather all the required background information and provide the details to the IBM Software Support for investigation.

Troubleshooting Web UI testing

This topic can help you troubleshoot some of the problems that you might encounter in Web UI testing.

Problem	Cause	Solution
At the start of recording a Web UI test using Microsoft Internet Explorer, the following error message pops up: `Error starting Internet Explorer.`	The browser is not properly installed on your machine, or you are using a version that is not supported by the Web UI extension, or the **Enable Protected Mode** option is not properly set.	• Check if the Microsoft Internet Explorer is properly installed on your machine and you are using a version supported by the Web UI extension.

Problem	Cause	Solution
		• Also, in the browser, go to **Tools > Internet Options > Security** tab and check if the **Enable Protected Mode** setting is the same for all the security zones.
You are unable to record and playback a Web UI test using Microsoft Internet Explorer. Typically, an object is highlighted in a blue rectangle during recording and in a red rectangle during playback.	The Web UI extension does not support testing web pages that are rendered in **Compatibility mode/Quirks mode** in Internet Explorer.	Perform the following steps to force websites to open in the standards view: 1. Open Internet Explorer. 2. Go to **Tools > Compatibility View Settings**. 3. Clear the following check boxes that force the web pages to be rendered in compatibility view: ◦ **Include updated website lists from Microsoft** ◦ **Display intranet sites in Compatibilty View** ◦ **Display all websites in Compatibility View**
Recording and playback of Web UI tests of a few websites using Microsoft Internet Explorer 11 fail with the following error: `Exception thrown: JavaScript error in async script.`	The website you are testing is not trusted.	Make sure you add the web sites to be tested to the list of trusted websites as follows: 1. Open Internet Explorer. 2. Go to **Tools > Internet Options > Security** tab. 3. Click **Trusted sites** and then click **Sites**. 4. Type the complete address of the website in **Add this website to the zone** and click **Add**. 5. Click **Close** and then click **OK**.

Problem	Cause	Solution
Playback of Web UI tests fails on Microsoft Internet Explorer with the following exception: `org.openqa.selenium.StaleElementReferenceException: Error setting arguments for script.`	Some websites use cookies to store information on your machine for automatically populating forms. On these sites, test scripts may have difficulty identifying controls based on the content, which results in playback failure.	Ensure that you delete all the cookies before playing back your test scripts.
Uninstalling Rational® Functional TesterHCL OneTest™ UI after recording/playing back tests with Microsoft Internet Explorer or Chrome does not automatically remove the `__SDP_PATH__` and `__BRAND_NAME__IMShared` folders. Manual removal of the folders also throws an error message.	The application fails to close the browser driver processes that are initiated as part of recording or playback.	Before you uninstallIBM® Rational® Functional TesterHCL OneTest™ UI, ensure that you manually kill any active driver processes associated with the browsers through the Windows Task Manager.
Playback fails on the submenu actions that are beneath the overlaid control.	If an application has a menu with a control overlaid on top and submenus beneath the overlaid control, the recorder might capture the action only on the overlaid control based on the application behavior. In that case, the submenus are not played back automatically.	You must manually add the submenus to the test steps for the playback to run successfully.

Testing mobile applications

You can test the Android or iOS mobile applications by recording and playing back mobile tests on Android devices or emulators, iOS mobile devices or simulators by using Rational® Functional TesterHCL OneTest™ UI.

You can find the following information about mobile testing:

- Testing Android applications on page 688
- Testing iOS applications on page 697
- Editing a mobile test on page 704

Testing Android applications

You can automate the testing of Android mobile applications in Rational® Functional TesterHCL OneTest™ UI by recording and playing back the recorded mobile tests.

Prerequisite tasks for recording Android mobile applications

Before you can use Rational® Functional TesterHCL OneTest™ UI to record a test for an Android application by using either an Android device or emulator, you must complete the following prerequisite tasks:

- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:
  - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
  - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.
- Connected and started the Android device that you want to use for testing Android mobile applications. You can connect the Android device to the computer through a USB cable.
- Installed the Android application that you want to test on the Android device.
- Installed Google Chrome browser on the computer where Rational® Functional TesterHCL OneTest™ UI is installed.
- Configured Android applications in a common web interface. See Configuring Android applications for mobile tests on page 690.

Recording tests for Android mobile applications

After you connect the Android device or emulator to your computer, you can then record the user interface and hardware actions in the Android application that you want to test.

You can perform the actions that you want to record by using a virtual client of the mobile device. The actions that you perform are captured as test steps in the test recording. You can perform the following actions in the test steps of a mobile test recording:

- Insert user actions.
- Insert navigation actions.
- Assign variables.
- Insert verification points.
- Substitute data by using custom code or datasets.

You can create a single test or multiple tests by recording actions that you perform for different functions on the mobile application.

See Recording mobile tests for Android applications on page 692.

You can add multiple tests to a Compound Test. You can also include the test variables or a variables file to the Compound Test.

See Creating a compound test on page 738.

You can create an Accelerated Functional Test Suite (AFT) Suite for mobile tests in the following scenarios:

- When you want to run a single mobile test on multiple devices.
- When you want to run multiple mobile tests on a single device.
- When you want to run multiple mobile tests on multiple devices that are connected on any of the following computers or device clouds:
    - Computer that runs Rational® Functional TesterHCL OneTest™ UI.
    - Remote agent computer.
    - BitBar Cloud.
    - BrowserStack Cloud.
    - Perfecto Mobile Cloud.
    - pCloudy Cloud.

See Creating an AFT suite for mobile tests on page 749.

Prerequisites for running tests for Android mobile applications

Before you can play back the recorded mobile test, you must complete the following tasks:

- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:
    - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
    - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
    - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.
- Connected and started the Android device that you want to use for testing Android mobile applications. You can connect the Android device to the computer through a USB cable.
- Installed the Android application that you want to test on the Android device.

Running tests recorded for Android mobile applications

By using Rational® Functional TesterHCL OneTest™ UI, you can play back the recorded mobile test on Android devices or emulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.

- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

You can play back a mobile test, multiple mobile tests, a compound test, or an AFT Suite by using Rational® Functional TesterHCL OneTest™ UI. After you select the test, you can select the location where the Android devices or emulators are connected and then specify the Android device or emulator on which you want to run the test.

When you play back the Compound Test containing mobile tests, you can specify different Android devices or emulators for each of the mobile test in the Compound Test.

See Configuration of mobile test runs on page 1277.

Viewing test results and reports

The test result is displayed as a unified report for the playing back of a mobile test. See Unified reports on page 1455.

Troubleshooting issues when testing Android applications

If you encounter any issues when you are testing Android applications, you can refer to the problems and their resolutions in Troubleshooting issues when testing Android applications on page 695.

Related reference

Task flows for testing mobile applications on page 41

Configuring Android applications for mobile tests

You can configure Android applications in a common web interface and use the configured applications any time later to record mobile tests. You can record mobile tests to perform actions on Android applications and automate the mobile tests.

1. Go to the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI.
2. Click the **Application configuration**  icon in the toolbar.
   The **Application configuration** page opens in the default browser.
   This page displays the list of applications that you configured in Rational® Functional TesterHCL OneTest™ UI.
3. Click **Add** and select **Android**.

   The **Android application** dialog box is displayed.
4. Choose a mode for configuring the Android application. You can click one of the following modes:

- **Manually**: Click this option to manually enter the complete details of the Android application.
- **From APK**: Click this option to either drag and drop the package file `.apk` or browse to select the package file of the Android application. After you drag and drop the package file in the **Drop APK or click to browse** field, the details of the Android application are populated automatically.
- **From device**: Click this option to select the Android application from devices or emulators that are connected to your computer.

5. Complete the action for each option as described in the following table:

Option	Action
Device **Note:** This field is displayed only when you select the **From device** mode.	Select a device or an emulator that is connected to your computer.
Package **Note:** This field is displayed only when you select the **Manually** or **From device** mode.	Perform one of the following actions based on the mode that you chose to configure the application: ◦ **Manually**: Enter the package name of the Android application that you want to configure. ◦ **From device**: Select the package name from the **Device** list. For example, the package name for YouTube is *com.google.Android.youtube*. You can also add an icon to the Android application by clicking the **Edit** icon inline with the **Package** field. This icon is prefixed to the application name.
Name	The application name is auto populated for **From APK** or **From device** modes. You can also edit the name as required. For the **Manually** mode, enter a name for the Android application with which you want to identify the application. This name is displayed as the **Application Name** after you configure the application. For example, the application name can be *add_forms*.
Activity	The activity name is used to identify the purpose of the Android application that you want to configure.

Option	Action
	After you enter the package name in the **Package** field, the same name is auto populated in the **Activity** field and it is suffixed with the text *.Main Activity*. For example, if the package name is *com.google.Android.youtube*, then the **Activity** name is auto populated as *com.google.Android.youtube.MainActivity*. You can edit the text as required.
Version	Enter a number that you want to identify the version of the mobile application used for the test. For example, the version number can be *dev_bld_-V123*.
Description	Enter a description for the application. For example, the application description can be *Test add video in bld V123*.

6. Click **Add**.

Results

You have configured the Android application.

What to do next

You can use the Android application to record the mobile tests.

Related information

https://developer.Android.com/studio/build/application-id

https://developer.Android.com/guide/components/activities/intro-activities

Recording mobile tests for Android applications

You can record a mobile test to capture the actions that you perform on Android applications. The actions are captured as test steps.

Before you begin

You must have completed the following tasks:

- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on any of the following operating systems:
  - Windows, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
- Connected the devices through a USB cable and configured the emulators or simulators on the computer that runs Rational® Functional TesterHCL OneTest™ UI.

About this task

Generally, when you record actions on the Android application, a control on the application is identified by using one of the control properties which is a unique identifier for that control. The following controls on the application can be identified by using the **Label** property also:

- Input field
- Drop-down list

For example, if there are 3 input fields on the application screen, then these input fields are identified by using the **Label** property which is unique to these input fields.

Restriction: In some Android devices, the double tap action that you perform on the lower part of the screen is not captured while you record the test.

1. Go to the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI.
2. Click the **New Test from Recording** icon in the toolbar and select **Mobile Test**.

   The **New Mobile Test From Recording** dialog box is displayed.
3. Select a directory to save the test.
4. Enter the name of the test in the **Test name** field, and click **Next**.
   The **Select mobile application** page displays the list of all mobile applications that you configured in the **Application Configuration** page.
5. Select the Android application that you want to test.

   Note: The real devices or the emulators that are connected to the computer are listed and you can select any device or emulator from the list.
6. Select the device from the **Select mobile** device list.
7. Click **Next**, and then click **Finish**.

   The application is displayed in a virtual client of both, the real mobile device or the emulator.

   The recording is started in a web browser.

You can perform the actions on the selected mobile application by using the virtual client. The actions that you perform are imitated on the device or emulator and are captured as test steps in the test.

The test steps are displayed on the **Test Steps** pane of the virtual client window.

The following table lists the options of the actions that you can perform in the virtual client:

Option	Action when clicked
Stop recording 	Stops the recording and generates the mobile test recording
Screen lock 	Performs the lock screen action on the mobile device or emulator
Refresh 	Refreshes the virtual client if the actions that you perform on the virtual client is not synchronized with the mobile device
Volume up 	Increases the volume on the mobile device or emulator
Volume down 	Decreases the volume on the mobile device or emulator
Mute 	Mutes the mobile device or emulator
Send SMS 	Sends an SMS to the phone number that you specify in the **Send SMS** dialog box
Make call 	Calls to the phone number that you specify in the **Make a call** dialog box
Overview 	Displays the previously opened applications
Home 	Navigates to the home screen
Back 	Navigates to the previous page or window in the mobile application

8. Click **Stop recording**.
   The recording of the test is stopped and the browser window is closed. The **Test Generation** dialog box is displayed in the **UI Test** perspective.

Results

You have recorded a test for an Android mobile application.

What to do next

You can perform any of the following tasks:

- You can click **Open Test** in the **Test Generation** dialog to view the recorded test and edit the test, if you want to change or add the test steps or verification points. See Editing a mobile test on page 704.
- You can run the recorded test. See Running mobile tests for Android mobile applications on page 1279.

Troubleshooting issues when testing Android applications

You can find information about the issues or problems that you might face while you perform mobile tests for Android applications. Details about issues, their causes, and the resolutions that you can apply to fix the issues are described.

The troubleshooting issues are presented to you in the following table:

Problem	Description	Solution

When you start recording the test, the recording does not continue and the following error message is displayed:
`Screenshot cannot be captured for the recording because the security flag is set for the current view.`
`To continue with the recording, switch to the non-secured view on the device and refresh the screen.`
	Rational® Functional TesterHCL OneTest™ UI needs to capture the screens for recording the tests. For some Android applications, because the security flag is set, the screen cannot be captured.	You must switch to the non-secured view of the application and continue with the recording.

When you record a mobile test on any Android application, if there are there two controls with the same name, then the playback fails.	When the application screen has two controls with the same name, during the playback, the control that is on the active area of the screen is identified as the recorded control even though the other control was clicked during the recording.	You must specify the location details for each of the controls that have the same name. You must perform the following tasks:

Problem	Description	Solution
	For example, consider that an application has two **Login** buttons, one at the top of the page and another at the bottom of the page. While recording if you clicked the second **Login** button and the first **Login** button was not visible in the active area, during the playback, both the **Login** buttons are visible and the playback considers the first **Login** button.	1. Click the step for which you want to add the location details. The **User Action Details** pane is displayed on the right side. 2. Select **at the specified index** from the **Object location** list. 3. Enter a value in the **Index** field. For example, for the first **Login** button, you can enter the value as *1* and for the second **Login** button, you can enter the value as *2*. 4. Save and run the test.
When you receive an incoming call on the application screen, you are not able to click on any of the options that are displayed on the pop-up frame.	When you want to record an incoming call action, you can initiate a call to the device, and then you can respond to the call by clicking one of the options in the pop-up frame. You might not be able to respond to the call because when you click the options in the pop-up frame, the option might not be clicked and the controls below the frame are clicked.	You can add navigation actions to respond to the incoming call. You must perform the following tasks: 1. Click the step for which you want to add a navigation action. 2. Click **Insert > Navigation action**. 3. Select either **Accept-Call** or **Decline-call** in the **Object's action** list, and then specify the phone number in the **from** field. 4. Save and run the test.
When you play back a Web UI test in the Chrome browser that is installed on an Android device, the playback	The unified report displays the following error message: `Error in launching Chrome. The browser might not be installed or`	You must perform the following steps:

Problem	Description	Solution
does not start and the unified report displays an error message.	`you are using an unsupported version.` This issue is because the version of the Chrome browser on the Android device is different from the version of the Chrome driver.	1. Download the appropriate Chrome driver. 2. Provide the path and the file name of the Chrome driver in the environment variable **WEBDRIVER_CHROME_DRIVER_MOBILE**. 3. Go to the directory path `<installation directory>/node-js/appium-server`, and then restart the execution agent.

Testing iOS applications

You can test the native and hybrid iOS applications in Rational® Functional TesterHCL OneTest™ UI by recording and playing back mobile tests on computers that run on Mac operating systems.

The following table lists the tasks that you must perform to record and run tests:

Tasks	Description
Prerequisite tasks for recording iOS tests on page 698	You must complete the prerequisite tasks before you record a test for iOS mobile applications.
Configuring the iOS applications on page 700	You must first configure the iOS application in Rational® Functional TesterHCL OneTest™ UI to record a mobile test.
Recording mobile tests for iOS applications on page 701	After you install the Xcode and the command line tools for Xcode, you can connect the iOS device or simulator to your computer, you can then record the user interface and hardware actions in the iOS application that you want to test. You can create a single test or multiple tests by recording actions that you perform for different functions on the mobile application.

Tasks	Description
Creating a compound test on page 738	You can add multiple tests to a Compound Test. You can also include the test variables or a variables file to the Compound Test.
Creating an AFT suite for mobile tests on page 749	You can create AFT suites and play them back on the following computers or device clouds: • Computer that runs Rational® Functional TesterHCL OneTest™ UI. • Remote agent computer. • BitBar Cloud. • BrowserStack Cloud. • Perfecto Mobile Cloud. • pCloudy Cloud.
Running mobile tests for iOS mobile applications on page 1310	You can play back the recorded mobile test on iOS devices or simulators.
Running compound tests for iOS mobile applications on page 1323	You can play back compound tests on iOS devices or simulators.
Viewing Unified reports on page 1455	You can view the test result that is displayed as a unified report after the playing back of a mobile test.
Troubleshooting issues when testing iOS applications on page 703	If you encounter any issues when you are testing iOS applications, you can refer to the problems and their resolutions.

Related reference

Task flows for testing mobile applications on page 41

Prerequisite tasks for recording iOS tests

You can test the native and hybrid iOS applications in Rational® Functional TesterHCL OneTest™ UI by recording and playing back mobile tests on computers that run on Mac operating systems.

Prerequisite tasks for recording iOS mobile applications on a simulator

Before you can use Rational® Functional TesterHCL OneTest™ UI to record a test for an iOS application by using a simulator, you must complete the following tasks:

- Set up a computer that runs on the Mac operating system.
- Installed Xcode and Command Line Tools for Xcode on your computer.

Note: You might have to follow certain steps in setting up the project with Xcode if you experience any issue while you record or play back iOS tests. See Troubleshooting issues when testing iOS applications on page 703.

- Installed the iOS applications that you want to test on the simulator.
- Configured the iOS applications that you want to test in Rational® Functional TesterHCL OneTest™ UI. See Configuring the iOS applications on page 700.

Prerequisite tasks for recording iOS mobile applications on an iOS device

Before you can use Rational® Functional TesterHCL OneTest™ UI to record a test for an iOS application by using an iOS device, you must complete the following tasks:

- Set up a computer that runs on the Mac operating system.
- Installed Xcode and Command Line Tools for Xcode on your computer.

Note: You might have to follow certain steps in setting up the project with Xcode if you experience any issue while you record or play back iOS tests. See Troubleshooting issues when testing iOS applications on page 703.

- Signed in to the WebDriverAgent Xcode project by using the Apple developer account.

Note: You must navigate to the `<installation directory>/node-js/appium-server/node_modules/appium/node_modules/appium-webdriveragent` folder and then follow the steps as described in the Setting up iOS Real Devices Tests with XCUITest portal.

- Installed the iOS applications that you want to test on the iOS device.
- Configured the iOS applications that you want to test in Rational® Functional TesterHCL OneTest™ UI. See Configuring the iOS applications on page 700.
- Connected and started the iOS device that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.
- Provided the Apple Team ID and role details in the **Apple Team ID** and **Role** fields.

Note: To enter the details, you must go to **Windows > Preferences > Test Execution > UI Test Playback > Mobile Device** tab.

- Added the iOS real devices to the same developer profile.

Related reference

Task flows for testing mobile applications on page 41

Configuring the iOS applications

In Rational® Functional TesterHCL OneTest™ UI, you can test the native and hybrid iOS applications on Mac operating systems. To test an iOS application, you must first configure the iOS application.

1. Go to the **Web UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI.
2. Click **Application Configuration**  in the toolbar.
   The **Application Configuration** window is displayed. This window displays the list of applications that you have configured in Rational® Functional TesterHCL OneTest™ UI.
3. Click **Add** and select **iOS**.

   The **iOS application** dialog box is displayed.
4. Choose a mode for configuring the iOS application. You can click one of the following modes:
   - **Manually**: Click this option to manually enter the complete details of the iOS application.
   - **From IPA**: Click this option to either drag and drop the package file `.ipa` or browse to select the package file of the iOS application. After you drag and drop the package file in the **Drop IPA or click to browse** field, the details of the iOS application are populated automatically.
5. Complete the action for each option described in the following table:

Field	Action
Bundle ID	Enter the bundle ID of the iOS application that you want to test. Each iOS application contains a unique bundle ID.
Name	Enter the name of the application that you want to configure. For example, *FormApp*.
Version	Enter a number based on which you want to identify the version of the mobile application used for the test. For example, the version number can be *dev_bld_V102*.
Description	Enter a description for the application.

Field	Action
	For example, the application description can be *Test add video in bld V102*.

6. Click **Add**.

Results

The iOS application is configured successfully.

What to do next

You can use the iOS application to record the mobile tests.

Recording mobile tests for iOS applications

You can record a mobile test to capture the actions that you perform on iOS applications. The actions are captured as test steps to a mobile test recording.

Before you begin

You must have completed the following tasks:

- Read and completed the prerequiste tasks listed in Prerequisite tasks for recording iOS tests on page 698.
- Ran the UI Test Agent on the default port to connect to your mobile devices or simulators.

About this task

After you install the Xcode and the command line tools for Xcode, you can connect the iOS device or simulator to your computer, you can then record the user interface and hardware actions in the iOS application that you want to test.

You can perform the actions that you want to record by using a virtual client of the mobile device. The actions that you perform are captured as test steps in the test recording. You can perform the following actions in the test steps of a mobile test recording:

- Insert user actions.
- Insert navigation actions.
- Assign variables.
- Insert verification points.
- Substitute data by using custom code or datasets.

Generally, when you record actions on the iOS application, the controls on the application are identified by using one of the control properties which is a unique identifier for that control. The input fields on the iOS applications are identified by using the **Label** property also.

If the port number of the computer on which you run the UI Test Agent is not the default port, then you must specify the port number. To specify the port number, you must perform the following tasks:

1. Click **Windows > Preferences > Test Execution > UI Test Playback > Mobile Device** tab.
2. Select the**Host** check box.
3. Enter the port number in the **Port** field.

You can then perform the following steps to record a mobile test on the iOS application.

1. Go to the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI.
2. Click the **New Test from Recording** icon in the toolbar and select **Mobile Test**.

   The **New Mobile Test From Recording** dialog box is displayed.
3. Select a directory to save the test.
4. Enter the name of the test in the **Test name** field, and click **Next**.
   The **Select mobile application** page displays the list of all mobile applications that you configured in the **Application Configuration** page.
5. Select the iOS application that you want to test.

   Note: The real devices or the simulators that are connected to the computer are listed and you can select any device or simulator from the list.
6. Select the device from the **Select mobile** device list.
7. Click **Next**, and then click **Finish**.

   The application is displayed in a virtual client of both, the real mobile device or the emulator.

   The recording is started in a web browser.

   You can perform the actions on the selected mobile application by using the virtual client. The actions that you perform are imitated on the device or emulator and are captured as test steps in the test.

   The test steps are displayed on the **Test Steps** pane of the virtual client window.

   The following table lists the options of the actions that you can perform in the virtual client:

Option	Action when clicked
Stop recording	Stops the recording and generates the mobile test recording
Screen lock	Performs the lock screen action on the mobile device or simulator

Option	Action when clicked
Refresh 	Refreshes the virtual client if the actions that you perform on the virtual client is not synchronized with the mobile device
Volume up 	Increases the volume on the mobile device or simulator
Volume down 	Decreases the volume on the mobile device or simulator
Home 	Navigates to the home screen

8. Click **Stop recording**.
   The recording of the test is stopped and the browser window is closed. The **Test Generation** dialog box is displayed in the **UI Test** perspective.

Results

You have recorded a test for an iOS mobile application.

What to do next

You can perform any of the following tasks:

- You can click **Open Test** in the **Test Generation** dialog to view the recorded test and edit the test, if you want to change or add the test steps or verification points. See Editing a mobile test on page 704.
- You can run the recorded test. See Running mobile tests for iOS mobile applications on page 1310.

Troubleshooting issues when testing iOS applications

You can find information about the issues or problems that you might face while you perform mobile tests for iOS applications. Details about issues, their causes, and the resolutions that you can apply to fix the issues are described.

The troubleshooting issues are presented to you in the following table:

Problem	Description	Solution
The UI Test Agent log displays the following message either when the recording or playback of iOS test does not start: `Unable to launch WebDriverAgent because of xcodebuild`	When you start to either record or play back the iOS test, the action does not proceed and the UI Test Agent log displays a message about the XCode failure.	You must perform the following steps; 1. Go to the location `<installation directory path>/node-js/ap-`

Problem	Description	Solution
`failure: xcodebuild failed with code 65 xcodebuild error`		`pium-server/node_modules/appium/node_modules/appium-webdriver-agent`, and then open `WebDriverAgent.xcodeproj`. 2. Select **WebDriverAgent** in the Project. 3. Click the **Build Settings** tab. 4. Click **All** and **Combined** in the row next to the **Build Settings** tab. 5. Set **Validate Workspace** value to *yes* in the **Build Options (i-OS)**section.

Editing a mobile test

You can edit a mobile test by using the test editor. You can edit a mobile test by modifying the test steps, adding verification points, and so on.

The test scripts are displayed in the test editor. The test steps are generated as simplified test scripts in the form of English statements which are easy to understand. The test steps display the list of keyboard actions and UI elements that were captured during the recording phase.

Note: Editing of a mobile test is similar to editing a Web UI test. If you are familiar with the recording, editing, and playing back the Web UI tests, it is easy for you to test a mobile application in Rational® Functional TesterHCL OneTest™ UI. See Editing Web UI tests on page 573.

The test editor has multiple views that are explained in the following table:

View name	Description
Test Contents view	The area at the top left is the **Test Contents** view. This view displays the chronological sequence of events in the test.
User Action Details view	The area at the top right is the **User Action Details** view. This view displays details about the currently selected action in the test script. In this area, you can select a graphic object and specify the action related

View name	Description
	to the object, specify how the object is identified and its location, and other such details.
SmartShot View	The area at the bottom left is the **SmartShot View**, which includes two tabs: **SmartShot** and **Elements**. The **SmartShot** tab displays the graphical objects captured during the recording of each test step. To the right is the **Properties** view, which shows the properties of the selected object.

You can use the **SmartShot View** to perform the following actions on the mobile test:

Actions on the mobile test	More information
Modifying a test step	Modifying a step in a test from the SmartShot View on page 597
Adding verification points	Creating verification points for objects in tests on page 580
Adding a loop	Adding a loop on page 587
Defining variables with a selected mobile device	Declaring test variables to select a mobile device on page 705
Assigning variables to an object property	Assigning a test variable to an object property on page 585
Adding user actions in a test	Adding user actions in a test
Applying the guided healing feature	Overview of guided healing and self-healing on page 617
Identifying the UI elements by using image property	Validating images and user interface elements by using the image property on page 624
Modularizing test scripts	Splitting UI actions on page 592
Inserting log messages into tests scripts	Inserting a custom log message into a UI test script on page 679

Declaring test variables to select a mobile device

To be able to launch subsequent tests in the same logical flow (session) from the same devices, you must select a variable including a reserved variable name and selection criteria related to one or multiple devices.

About this task

Declaring a test variable is applicable to Android, iOS, hybrid and Web UI applications.

1. In the Test Navigator, browse to the test and double-click it.
   Result
   The test opens.
2. To create a container for the test variables that you create in a test:
   a. Open the test, and in the **Test Contents** area, click **Test Variables**, at the top of the test.
   b. Select **Add > Test Variable Container**.
      Result
      A container named **Test Variables** is created for the user-defined variables.
   c. Select the container to rename it.
      Result
      The **Test Variables Details** area opens for you to type a new name in the **Name** field.
3. To declare a variable in a test:
   a. Open a test and select the test variable node.
   b. Click **Insert > Variable Declaration**
   c. Enter the name of the variable for the mobile device selection as `Mobile_Device_Selection`.
      The variable is automatically initialized to the name of the Android device or emulator where the recording of test was performed.
   d. Click **OK**. The variable is added as the last element in the container and the **Variable Details** area opens.
   e. In the **Visible in** section, select **This test only** to restrict data to the current test only. Even if another test has a variable with the same name, that variable will not change. Select **All tests for this user** to share the value of this variable when the test runs in a compound test. For the variable to be shared, both tests must have a variable with the same name and must have this option enabled.
4. Assign a specific value to the variable and initialize the variable:
   a. Select **Text**
   b. Enter a selection sentence to assign a variable value to a text string. Enter selection strings including a devices property, followed by an operator value, propertys value and a comma separating each string. For more details on the main device properties you can use and on the syntactic rules, see the Variable selection values on page 1700 topic.

Results

The variable can then be initialized from some external source (a dataset, tests from an IBM® Rational® Quality Manager test suite, or tests from the same user in compound tests containing one or more mobile tests. It can also be set within a tests execution with a variable assignment action from any data source, including a data correlation

reference, custom code, built-in function, dataset, or string constant. As a result, successive tests in the same session will then be sure to run on the same actual devices.

Note: When a test launches an application:

- The *Mobile_Device_Selection* variable content is checked to get the device ID
- The device is reused if it is still applicable to the app that must be launched. Conditions:
  - The device operating system must be the same as the operating system of the application to be launched.
  - The tester app is installed or can be installed without user intervention.

Testing Windows desktop applications

You can configure the Windows applications to record and play back tests on the Windows desktop applications. You can also test the Windows applications on dual monitors when you extend the display of your computer to a secondary monitor.

Rational® Functional TesterHCL OneTest™ UI supports the testing of the following applications:

- **Windows desktop applications**: The Windows desktop applications include two types of applications such as Universal Windows Platform (UWP) and Classic Windows applications.
- **Add-ins in MS Office**: The Microsoft Office applications such as Word, Outlook, Access, and PowerPoint are supported. Add-in applications which are integrated and available on the toolbar of the Office applications are also supported.

Prerequisites for testing Windows desktop applications

You can find information about the prerequisite conditions that you must satisfy before you can test Windows desktop applications.

Before you record and play back Windows tests, you must have completed the following actions:

- Configured the Windows desktop application. See Configuring a Windows application on page 708.
- Integrated Add-ins with the Microsoft Office application.
- Restarted your computer if you want to record the Windows test immediately after you install the product.

Tip: To run tests on your local computer, you must clear the **Host** checkbox in the **Mobile Device** tab. You can access the checkbox from the **Preferences > UI Test Playback > Mobile Device** tab.

Related information

Configuring a Windows application on page 708

Recording a Windows test on page 709

Editing a Windows test on page 713

Configuring a Windows application

In Rational® Functional TesterHCL OneTest™ UI, you can test the native Windows desktop applications. To test the Windows applications, you must first configure the Windows applications in the common web interface.

About this task

After you configure the Windows application in the common web interface, you can use the Windows application anytime later to record Windows tests.

1. Go to the **Web UI Test** perspective.
2. Click the **Application Configuration** icon on the toolbar.

   A browser window opens and the **Application Configuration** page is displayed. The page displays the list of all applications that you configured in Rational® Functional TesterHCL OneTest™ UI.
3. Click **Add** and select **Desktop**.

   The **Desktop Application** dialog box is displayed.
4. Select one of the following types of the Windows desktop application that you want to configure:
   - **EXE**: Click **EXE** to configure classic Windows desktop applications.
   - **UWP**: Click **UWP** to configure Universal Windows Platform (UWP) applications.
5. Enter the following detail based on the type of the Windows application:
   - **Exe path**: Enter the path that contains the `.exe` file of the classic Windows application on your computer.
   - **Application ID**: Enter the application id of the Windows application. For example, if you want to configure alarm app, you must enter *Microsoft.WindowsAlarms_8wekyb3d8bbwe!App*.

     Note: To get the id of the Windows application, you must run the Windows application, and then click **Properties** to copy the application id.
6. Enter a name for the Windows application in the **Name** field.
7. Enter the version number for the Windows application in the **Version** field.

8. Enter details about the Windows application in the **Description** field.
9. Click **Advanced Options** to enter more details about the Windows application.

Results

You have successfully configured a Windows application.

What to do next

You can record tests on the configured Windows application. See, Recording a Windows test on page 709.

Recording a Windows test

You can record a Windows test to capture the actions that you perform on Windows desktop applications and on the Add-ins in Microsoft Office applications. The actions are captured as test steps in Windows tests.

Before you begin

You must have completed the tasks mentioned in Prerequisites for testing Windows desktop applications on page 707.

About this task

Generally, when you record actions on the Windows desktop application, the controls on the application are identified by using one of the control properties which is a unique identifier for that control. For some of the controls on the Windows applications, when the **Name** property is also available the **Label** property is not generated. The **Label** property is generated only for the following controls when the **Name** property is not available:

- Input field
- List
- Combo box

You can record actions on multiple windows including the dialog boxes of the Windows desktop application. You can also press Alt+Tab to navigate between the child and the main windows while recording the Windows test.

Tip:

- When you record a test on the Windows application, you must wait when the cursor is busy, and then proceed when the cursor changes to the default state.
- When the Add-in application response time is more than expected, you can press the left Shift key twice to synchronize the application state with Rational® Functional TesterHCL OneTest™ UI. The cursor is then is restored to the default state.

  Before you can perform the next action on the Add-in application, you must click on the Add-in application once.

1. In the **UI Test perspective**, click **New > Test From Recording**.

   The **Windows Test Recorder** is displayed.

   Note: Alternatively, you can click the **New Test From Recording** icon on the toolbar and click **Windows Test**.

2. Click **Create a test from a new recording** and then select **Windows Test**.
3. Select a project under which you want to save the Windows test, and then enter a name for the test that you want to record.
4. Click **Next**.

   The **Select Windows application** dialog box displays all the configured Windows applications.

   Note: An error message is displayed if the WinAppDriver is not running. You must run the WinAppDriver.

5. Select a Windows application from the list, and then click **Finish**.

   The Windows application is displayed.

   Note: When you record a test on a Windows application, you must ensure that the Windows application does not overlap with the recording tool bar. Otherwise, the recording tool bar is also captured in the recording.

6. Perform the necessary actions on the Windows application to record the test.

   Tip: To capture all the steps in the test result, you must be slow while performing actions on the Windows application.

7. Stop the recording. To stop the recording, you must close the Windows application and then click the **Stop** icon in the Recording Control view.

Result

The **Test Generation** dialog box is displayed.

Results

You have successfully recorded a test on the Windows application.

What to do next

You can edit the generated Windows test in the test editor. For information, see Editing a Windows test on page 713.

Supported Windows UI controls

Rational® Functional TesterHCL OneTest™ UI supports the testing of various standard Windows UI controls.

The following Windows UI controls are supported for a Windows test:

- Window
- Pane
- Title Bar
- Menu Item
- Image
- Button
- Drop-down list
- List Item
- Group
- Text
- Scroll Bar
- Hyperlink
- Edit
- Tab
- Tab view
- Thumb
- Combo box
- Menu bar
- Tree
- Tree item
- Radio button
- Check box
- Status bar
- Slider
- Document
- Table
- Header

- Header item
- Data item
- Custom

Note: When you record Windows tests, although you can successfully capture the different types of drop-down lists or combo boxes, certain list controls might not be captured.

The list controls, which are spanned from the application under test, do not exist as part of the application and they are not captured correctly.

Supported keyboard and mouse actions

Rational® Functional TesterHCL OneTest™ UI supports the basic keyboard and mouse actions that you can perform while recording a Windows test.

You can record actions on certain controls by performing the following tasks:

To record the action	You must...
Hover over a control	Move the cursor over the control, and then press the left **Shift** key once.
Right-click a control	Move the cursor over the control, and then right-click the control.
Drag a control	Move the cursor over the control, and then drag the control.
Double-click a control	Move the cursor over the control, and then double-click the control.

Tip: During the recording of any Windows application, for certain context menus with multiple submenus or dialog boxes, the actions that you perform on these controls might not be recorded correctly. To overcome this issue, you can manually add the navigation actions such as presskey and inputkey to the generated test. You can provide the keyboard strokes to the navigation actions. For more information about the navigation actions, see Simulating keyboards and special keys actions on Web and native application windows on page 576.

The following keyboard keys are supported for a Windows test:

- Keypress
- Ctrl+Shift+<char>
- Ctrl+Tab
- Shift+<char>
- Alt+Tab

- Ctrl+P
- Ctrl+N
- F1-F12 (with special keys)
- PgUp
- PgDn
- Enter
- Esc
- Arrow keys (Up/Down/Left/Right keys with special keys)
- Home/End/Insert/Delete (with special keys)

Editing a Windows test

You can edit the test by using the test editor. You can edit a Windows test by modifying the test steps, adding verification points, and so on.

The test scripts are displayed in the test editor. The test steps are generated as simplified test scripts in the form of English statements which are easy to understand. The test steps display the list of keyboard actions and UI elements that were captured during the recording phase.

Note: The Windows test is similar to a Web UI test. If you are familiar with the recording, editing, and playing back the Web UI tests, it is easy for you to test a Windows application in Rational® Functional TesterHCL OneTest™ UI. See Editing Web UI tests on page 573.

The test editor has multiple views that are explained in the following table:

View name	Description
Test Contents view	The area at the top left is the **Test Contents** view. This view displays the chronological sequence of events in the test.
User Action Details view	The area at the top right is the **User Action Details** view. This view displays details about the currently selected action in the test script. In this area, you can select a graphic object and specify the action related to the object, specify how the object is identified and its location, and other such details.
SmartShot View	The area at the bottom left is the **SmartShot View**, which includes two tabs: SmartShot and Elements. The **SmartShot** tab displays the graphical objects captured during the recording of each test step. To

View name	Description
	the right is the **Properties** view, which shows the properties of the selected object.

You can use the **SmartShot View** to perform the following actions on the Windows test:

Actions on the Windows test	More information
Modifying a test step	Modifying a step in a test from the SmartShot View on page 597
Adding verification points	Creating verification points for objects in tests on page 580
Adding a loop	Adding a loop on page 587
Assigning variables to an object property	Assigning a test variable to an object property on page 585
Adding user actions in a test	Adding user actions in a test
Applying the guided healing feature	Overview of guided healing and self-healing on page 617
Identifying the UI elements by using image property	Validating images and user interface elements by using the image property on page 624
Modularizing test scripts	Splitting UI actions on page 592
Inserting log messages into tests scripts	Inserting a custom log message into a UI test script on page 679

Recording SAP tests

When you record a test, the test creation wizard records your interactions with the SAP server, generates a test from the recording, and opens the test for editing. You can record tests from the SAP GUI. You can also record SAP batch input tests that can be used to produce a heavy load on the server while minimizing the processing requirements for virtual testers.

SAP testing guidelines

Before you test the SAP applications, you must set up your test environment and incorporate these guidelines to produce reliable SAP tests.

SAP configuration

The SAP GUI client software must be installed on the same computer as IBM® Rational® Functional TesterHCL OneTest™ UI. The SAP GUI client is required for recording and running tests. For information about support of SAP GUI versions, refer to the *SAP Note 1412821 - SAP GUI for Windows: Support on Windows* for SAP.

The product is optimized by default for SAP GUI version 7.10 or later. To improve performance with older SAP GUI 6.20 and 6.40 versions, when running long tests, you must change the `bridge2java.dll` file located in the `C:\Program Files\IBM\SDP\plugins\com.ibm.rational.test.lt.runtime.sap\<build_identifier>C:\Program Files\HCL\HCLOneTest\plugins\com.ibm.rational.test.lt.runtime.sap\<build_identifier>` directory by renaming `bridge2java.dll` to `bridge2javaV7.dll`, and then renaming `bridge2javaV6.dll` to `bridge2java.dll`.

If you are deploying tests on remote computers to simulate a large number of users, the following software must be installed on each remote computer:

- The SAP GUI client software, configured with the same logon properties as the client on which the tests were recorded
- The IBM® Rational® Functional TesterHCL OneTest™ UI software that is provided with the product

Testing relies on the SAP Scripting API and ActiveX. Make sure that these options are selected when installing the SAP GUI client.

Test recording and running also require that scripting be enabled on the SAP application server and on all SAP GUI clients that are installed on remote computers. See the topic on configuring SAP for testing for more information.

Limitations

During playback of the tests, each virtual user runs SAP GUI in silent mode by default (the user interface is not displayed on the screen). However, some modal dialog boxes from the SAP GUI might briefly flash on the screen.

Avoid recording SAP tests with the SAP GUI low speed connection setting. You cannot run with a normal speed connection tests that you recorded with this setting.

You can have a maximum of only 50 virtual users on an agent for a SAP GUI test.

Batch input tests

You can use batch input tests to simulate a large number of virtual users while minimizing the load on the virtual user computers.

Batch input tests access the SAP server at a low level, bypassing the SAP GUI interface, and therefore cannot contain any verification points or SAP GUI elements. Their main purpose is to simulate a load on the server when added to a test that already contains SAP tests. Only the SAP tests will provide accurate SAP application performance measurement.

Batch input transactions are recorded in the SAP GUI and exported to the file system. You can then generate batch input tests that are based on those recorded transactions.

Note: Sometimes the default values of the SAP Java Connectors (JCo) parameters are not sufficient for the load tests. The default values require some updates both at the SAP R/3 server and client end. For the client, in the **Additional SAP Connection Properties** window, you must configure the SAP JCo parameters options that are available in the test. Alternatively, you can specify the properties as RPT_VMARGS in the agent location. For example, RPT_VMARGS=- Djco.cpic_maxconv=1000

Long duration test runs

When running long duration test schedules that exceed many hours, use the long run mode process to reduce resource consumption with the SAP GUI client. This mode increases the reliability of long duration test runs because a new process starts every time the specified number of SAP sessions completes. Hence, Windows resource consumption is bare minimum. You must enable this option for each user group in the schedule.

In the long run mode, a new process is created each time the number of SAP test instances reaches the specified number.

Cleaning the SAP work directory

In some cases, trace files are created by SAP GUI under `SapWorkDir` directory when running SAP tests.
You can delete these files by setting and environment variable *RPT_CLEAN_SAPWORKDIR* or a java VMARG *rptCleanSapWorkDir*. For example:

- `-DrptCleanSapWorkDir=C:\Documents and Settings\UserName\SapWorkDir`
- `RPT_CLEAN_SAPWORKDIR="C:\Users\UserName\AppData\Local\SAP\SAP GUI\Traces"`

If you set the variable to the `SapWorkDir` folder location, the contents (*.trc files) of the folder are removed when a schedule starts. If the variable is set to `true` or `on`, the product automatically searches for the `SapWorkDir` folder before removing its contents. If the variable is set to `false` or `off`, no action is taken.

Recording an SAP test

You can record your interaction with the SAP GUI client to generate an SAP test. When you record, the recording wizard opens the SAP GUI client and records all the interactions that occur between the client and the server.

Before you begin

You must have completed the following tasks:

- Verified that SAP GUI scripting is enabled on the SAP server and the SAP GUI client.
- Working SAP GUI client that you can connect to an SAP server.

- Verified that tests are stored in test projects. If your workspace does not contain a project, the test creation wizard enables you to creates one.
- Ensured that the session that you are recording is reproducible.

  For example, if you create items in SAP and do not delete them, then if items created in SAP are already exist when the test is run, that might cause the test to not run as expected.

About this task

You must install the latest version of SAP GUI when you want to record an SAP test in Rational® Functional TesterHCL OneTest™ UI.

Important:

Both 64-bit and 32-bit Rational® Functional TesterHCL OneTest™ UI supports the SAP protocol.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click the **New Test From Recording** icon, and then click **SAP Test**.

   Note: You can also click **File > New > Test From Recording**, and then select **SAP Test**. You can then select the encryption level, if required and click **Next** to open the **Select Location** page.

   Result

   The **Select Location** page is displayed.
3. Create a test by performing the following steps:
   a. Select a project, and then select a folder from the project.
   b. Enter a name for the test.
   c. Optionally, click **Recording encryption level**, and then select the encryption level when you are recording any sensitive data.
4. Click **Next**.
   Result

   The **Select Client Application** page is displayed.
5. Select **SAP Batch Input Recording** or **SAPGUI For Windows**, and then click **Next**.

   Note: When you record an SAP test by using the **SAP GUI For Windows** method and if you use many split points, then steps might not split properly after the SAP recording is complete. The split action during the SAP recording comes into effect only after an SAP request by changing the state of the

current screen. After the test generation is complete, you must use the **Split Test** action for splitting the steps to different SAP tests from the test editor.

6. On the **SAP Connection** page, select how to connect to the SAP server:
   Choose from:
   - In most cases, select **SAP Logon**; then enter the description normally used by SAP Logon to identify the server in **SAP system name**.
   - If your environment does not support SAP Logon, select **Server information**. In **Application server**, enter the host name or IP address of the server. Then specify a value for **System number**. Enter information in **Other options**, if required. Refer to your SAP documentation for details about the other SAP Logon options.
   - If your environment uses gateways or routers to connect to the SAP server, select **Connection by string**. Click **Edit** to specify a valid connection string. Refer to your SAP documentation for details about connection strings.
   - If you have an SAP shortcut file to automate the connection, select **SAP shortcut file**. Click **Browse** to specify the location of the file.
7. If this is the first time you record a SAP test, read the privacy warning, and select **Accept**.
8. Click **Finish** to start recording.
   Result
   In some cases, you might see a warning that a script is opening a connection to SAP.
9. Log on to SAP and complete the transactions to test.
   For security reasons, the password cannot be recorded by the SAP test recorder. Instead, it is requested at the end of the recording session.
10. In the SAP GUI window, perform the tasks for testing.
    You can use the **Recorder Test Annotations** toolbar to add comments, record synchronizations, or take screen captures during the recording.
    - To add a comment to the recorded test, click the **Insert comment** icon .
    - To add a screen capture to the recorded test, click the **Capture screen** icon . Screen and window captures make your tests easier to read and help you visualize the recorded test. You can change the settings for screen captures and add a comment to the image.
    - To manually add a test synchronization to the recording, click the **Insert synchronization** icon .
    - To insert a split point into the recorded test, click the **Split point** icon button. Split points allow. With split points, you can generate multiple tests from a single recording, that you can replay in a different order with a schedule.
11. When you have completed the transactions to be tested, stop the recorder by closing the SAP GUI or by clicking **Stop** in the **Recorder Control** view.
12. In the **Enter Password** window, enter the password for the account that was used for recording.
    This step is required because SAP GUI does not allow direct recording of the password.

Result

A **Test Generation Progress** window is displayed while the test is being generated.

The following message is displayed on the progress window: `Test Generation completed`.

What to do next

You can now play back the test and check the test results.

Before playing back a test, in SAP Connection Details editor, click **Test Connection** to test the connection to the SAP GUI server.

Inserting a new recording into a SAP test

You can insert a new recording into a test. Use this feature to add or replace a part of a recorded session.

Before you begin

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.

You must install the latest version of SAP GUI when you want to record an SAP test in Rational® Functional TesterHCL OneTest™ UI.

1. 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.
2. Click **Insert**, and then **New recording**.
   Result
   The test starts replaying up to the selected insertion point.
3. When the **New Recording** window is displayed, perform the sequence of actions that you want to add to the existing test.
4. When you have finished, in the **New Recording** window, click **Stop** to stop the recording.
   Result
   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 contents.
5. After the test has been updated in the Test Navigator, check that the new sequence was properly inserted into the test, and then click **File > Save** to save the test or **File > Revert** to cancel the inserted recording.

Recording a SAP batch input test

You can record certain *SAP transaction* sessions from SAP GUI with SAP batch input tests recording wizard. When you record a session, the recording wizard automatically starts a SAP GUI interface and records the transaction that you specified. After you finish the recording, the wizard generates a SAP batch input test in IBM® Rational® Functional TesterHCL OneTest™ UI.

About this task

During a SAP batch input test recording, the SAP batch input test produces only a batch input transaction that you specified. The SAP batch input tests do not contain any verification point and do not produce any performance result.

You must install the latest version of SAP GUI when you want to record an SAP test in Rational® Functional TesterHCL OneTest™ UI.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click the **New Test From Recording** icon, and then click **SAP Test**.

   Note: You can also click **File > New > Test From Recording**, and then select **SAP Test**. You can then select the encryption level, if required and click **Next** to open the **Select Location** page.

   Result

   The **Select Location** page is displayed.
3. Create a test by performing the following steps:
   a. Select a project, and then select a folder from the project.
   b. Enter a name for the test.
   c. Optionally, click **Recording encryption level**, and then select the encryption level when you are recording any sensitive data.
4. Click **Next**.
   Result

   The **Select Client Application** page is displayed.
5. Select **SAP Batch Input Recording**, and then click **Next**.
6. Perform the following steps to enter the connection and transaction details provided by the SAP administrator.

   a. Enter the following details in the **Connection** section.

Fields	Action
Client	Enter the SAP client details.
User	Enter the user name associated with the SAP server.
Password	Enter the password associated with the user name.
Language	Select your language.
Host	Enter the SAP host server details.
System Number	Enter the system number that corresponds to an SAP instance with the SAP server.

Fields	Action
	Note: For IBM® Rational® Functional TesterHCL OneTest™ UI, the default value is **00**.

b. Click **Test Connections** to verify if the connection is established.
Result

The **SAP Batch Connection** dialog box is displayed.

A confirmation message is displayed that states that the connection is successful.

c. Click **OK**.

d. Enter the SAP transaction code in the **Code** field.

e. Click **Finish**.

Result

The **SAP GUI** window is displayed.

7. Record the batch input transaction, and then click **Save**.
Result

The **SAP GUI Transaction** page is displayed.

The page displays the details of the transaction that you recorded.

8. Click **Exit**.

Note: To exit the SAP GUI Transaction page, you must click **Exit**. If you exit the page by using any other methods, an exception error is displayed.

9. Select the directory where you want to export the recording on your local computer, and then click **Generate**.
Result

The **SAPGUI Security** dialog box is displayed.

The dialog box displays the file path for the recording and the directory where the recording is being stored.

10. Click **Allow** to confirm.
Result

A **Test Generation Progress** window is displayed while the test is being generated.

The following message is displayed on the progress window: `Test Generation completed.`

Results

The SAP batch input transaction is recorded.

What to do next

You can click **Open Test** to open the test in Rational® Functional TesterHCL OneTest™ UI test editor. You can then verify the SAP transaction details and save the test. Later, you can run the test and view the transaction details from the following pages:

- Test Log page
- Functional Test report page

Changing SAP test generation preferences

You can change preferences for SAP tests to control the generation of SAP tests. You can also modify the setting for the verification points, data correlation, and the display of SAP GUI during the test run.

1. Open Rational® Functional TesterHCL OneTest™ UI, and then click **Window > Preferences**.
2. Expand **Test > Test Generation**, and then select **SAP Test Generation**.
3. Perform the actions described in the following table from the **Automatic Generation** section:

Options	Actions
Use connection by string	Select this checkbox to connect the SAP tests by using the string launch method instead of the SAP Logon program.
Verification points for SAP screen title	Select this checkbox to generate the verification points on-screen titles with each SAP screen.
Verification points for SAP request response time threshold	Select this checkbox to generate verification points on the response time of the SAP server. **Note:** If the server response time is higher than the specified threshold value, then the test produces a failed verification point.
Calculate threshold from recorded (%)	Enter a value in this field for the response time threshold that is calculated when response time verification points are generated.

Options	Actions
	Note: The threshold value is calculated as a percentage of the actual response time that was measured during the recording. **Remember:** The default value of the response time threshold is 120% of the recorded response time.
Default request timeout [ms]	Enter a value in this field for a request to ping the server. **Note:** When the request is timed out, it cannot ping the server for that request.

Remember: The default option that is selected in the **Automatic generation** section is **Use connection by string**.

4. Perform the actions described in the following table from the **GUI on execution** section:

Options	Actions
Hide GUI during execution	Select this checkbox to hide all instances of the SAP GUI. **Tip:** If you run an SAP test by selecting the **Hide GUI during execution** and the test fails due to modal dialog boxes or pop-up windows in transactions, you must add a **Location** property. The name and value of the property must be as follows: `RFT_VMARGS = rftSapForceShowNone=true`
Show GUI during execution	Select this checkbox to show all instances of the SAP GUI. By selecting this option, you can monitor the test run.

Remember: The default option that is selected in the **GUI on execution** section is **Hide GUI during execution**.

Irrespective of what you have selected in the **GUI on execution** section, you can still change the behavior of the display of SAP GUI during the test run time.

Consider that you have recorded 100 SAP tests by using the default selection of an option in the **GUI on execution** section. Due to some changes in your requirements, you want to run SAP tests by displaying SAP GUI during the test run time. In such instances, you can select and right-click the required tests from the **Test Navigator** pane, and then hover on **SAP GUI mode on execution** to choose the following options:

- **Set show GUI during execution**: to show all instances of the SAP GUI.
- **Set hide GUI during execution**: to hide all instances of the SAP GUI.

5. Select the **Prompt me for password when generating test** checkbox to request a password at the end of the recording session.

   When you clear this checkbox, the password is recorded with an empty string.
6. Click **Apply and Close** to save the changes.

Results

You have changed preferences for SAP test generation.

What to do next

You can run the test. See Running a Web UI test on page 1243.

SAP test editor reference

In SAP testing, the test editor information is divided into nine categories. This section describes the fields in each category that can be edited manually.

SAP test details

In the test editor, the test element is the first element in the test suite. These settings apply to the entire test.

SAP options

Display SAP GUI on execution

During test execution, it might not be desirable to display the SAP GUI. 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.

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 users.

Common options

Datasets

Lists details about each dataset used by the test: the name of the dataset, the columns that are used, and the location in the test where the dataset column is referenced. Click the location to navigate there.

Add dataset

Adds a reference to a dataset that you want a test to use. Clicking this option is the same as clicking **Add > Dataset** with the test selected.

Remove

Removes the selected dataset. This option is not available if the dataset is in use.

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 server. These settings apply to the entire test.

SAP system name

This is the description normally used by SAP Logon to identify the server. If the **Connection by string** option is selected this field is ignored.

Connection by string

Select this option to use the connection string that was returned by the server when recording to connect to the server without referring to the SAP Logon program. This is safer when deploying the test on remote computers. Advanced users can edit the connection string if necessary. You can use data correlation to substitute this value.

Get SAP GUI session statistics

Select this option to record session statistics from the SAP GUI client in the test results. These results are displayed on the **User Load** page of the test report.

Use new visual design

Select this option to run tests with a visual design theme when using SAP GUI 7.0 or later. In most cases, it is best to leave this option disabled, which causes tests to run with the default SAP GUI visual design and avoids compatibility issues.

Use recorded visual design theme

If **Use new visual design** is selected, select this option to use the visual design theme that was used during the recording o

Use other visual design theme

If **Use new visual design** is selected, select this option to use a specific visual design theme. Ensure that the name is correct and that the visual design theme is installed on the test computer. Unexpected results might occur if you specify a visual design theme name that cannot be located on the test computer.

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.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

Title

This is the recorded name of the SAP screen. This field is read-only.

Do not measure performance on this screen

Select this option if you do not want to obtain response time results for the current SAP screen. Use this for SAP screens that are not meaningful for your test, such as the logon screen.

Optional screen

Select this option if you do not want to log an error when the current SAP screen is not displayed. Use this for SAP screens that are not always displayed.

Data Table

Summarizes data substitutions and substitution candidates in the SAP screen. Double-click a row to navigate to the location where a substitution or candidate occurs. To associate a dataset candidate with a dataset, click the row and then click **Dataset Variable**. To remove a dataset substitution, click the row and then click **Remove Substitution**.

Screen Title Verification Point

Enable Verification Point

When selected, the test verifies whether the SAP screen returns the value shown in the **Expected screen title** field. An error is reported in the test log if the screen title returned during the test does not match the expected title.

Expected screen title

This field allows you to specify the expected SAP screen title. By default, the expected title is the recorded title. The expected title can optionally be expressed as a regular expression.

Recorded screen title

This field displays the recorded title of the current SAP screen. This field is read-only.

Use Regular Expression

Select this option to express the expected title using the standard regular expression syntax.

SAP set details

In the test editor, SAP sets are located in SAP screen elements and describe a user input action in the SAP GUI client. These settings apply to the selected SAP set.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

Think Time

Specifies the programmatically-calculated time delay that is observed for users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

SAP Set

Property name

This is the description of the GUI object related to the current SAP set as it appears to the user in the SAP GUI. This field is read-only.

Value

This is the value entered by the user in the current SAP set. You can use data correlation to substitute this value.

SAP GUI Object Information

Name

This is the recorded name of the GUI object related to the current element. This is a read-only field.

Type

This is the recorded type of the GUI object related to the current element. This is a read-only field.

Identifier

This field displays the recorded identifier of the GUI object that is related to the current element.

You can modify the identifier of an SAP script object by selecting the **Use regular expression** checkbox. You can then insert a regular expression in a segment for the selected field separated by a forward slash (/). You can revert to the recorded value by using the **Restore Recorded Id** option.

You must be well-versed with the notion of SAP identifier that is used in the SAP GUI scripting so that you can use the regular expression effectively. For more information, refer to the SAP documentation.

SAP get details

In the test editor, SAP get events are located in SAP screen elements and provide a way to retrieve data from a SAP GUI object to implement verification points. These settings apply to the selected get event.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

Think Time

Specifies the programmatically-calculated time delay that is observed for users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

SAP Get

Property name

This is the description of the GUI object related to the current event as it appears to the user in the SAP GUI client. This field is read-only.

Value

This is the value recorded during the test or during the last execution. You can use data correlation to reference this value. This field is read-only.

Verification Point

Enable Verification Point

When selected, the test verifies whether the screen returns the value specified in **Expected Value**. An error is reported in the test log if the value returned during the test does not match the expected value.

Expected Value

This field enables you to specify the expected value for the get event. The expected value can optionally be expressed as a regular expression. You can use data correlation to substitute this value.

Use Regular Expression

Select this option to express the expected value using standard regular expression syntax.

SAP GUI Object Information

Name

This is the recorded name of the GUI object related to the current element. This is a read-only field.

Type

This is the recorded type of the GUI object related to the current element. This is a read-only field.

Identifier

This field displays the recorded identifier of the GUI object that is related to the current element.

You can modify the identifier of an SAP script object by selecting the **Use regular expression** checkbox. You can then insert a regular expression in a segment for the selected field separated by a forward slash (/). You can revert to the recorded value by using the **Restore Recorded Id** option.

You must be well-versed with the notion of SAP identifier that is used in the SAP GUI scripting so that you can use the regular expression effectively. For more information, refer to the SAP documentation.

SAP call details

In the test editor, SAP call elements are located in SAP screen elements and describe various recorded interactions with the SAP server. These settings apply to the selected SAP event.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

Think Time

Specifies the programmatically-calculated time delay that is observed for users. Think time is a statistical emulation of the amount of time actual users spend reading or thinking before performing an action.

SAP Call

Method name

This is the internal method call used by the SAP GUI client. This field is read-only.

Parameter

If the method uses parameters, one **Parameter** line is displayed for each parameter. Advanced users can modify these parameters. Refer to SAP documentation for more information about the parameters used by SAP GUI method calls. You can use data correlation to substitute this value.

Return

If the method returns a value, a **Return** line is displayed, which can be used for data correlation or for a verification point. The value displayed is not the actual return value, but only represents the type of the parameter, for example, `string` for a string type or `0` for an integer. Refer to SAP documentation for more information about the parameters used by SAP GUI method calls.

Verification Point

Enable Verification Point

When selected, the test verifies whether the **Return** value of the SAP call (if any) matches the value specified in **Expected Value**. An error is reported in the test log if the value returned during the test does not match the expected value.

Expected Value

This field enables you to specify the expected value for the call. The expected value can optionally be expressed as a regular expression. You can use data correlation to substitute this value.

Use Regular Expression

Select this option to express the expected value using standard regular expression syntax.

SAP GUI Object Information

Name

This is the recorded name of the GUI object related to the current element. This is a read-only field.

Type

This is the recorded type of the GUI object related to the current element. This is a read-only field.

Identifier

This field displays the recorded identifier of the GUI object that is related to the current element.

You can modify the identifier of an SAP script object by selecting the **Use regular expression** checkbox. You can then insert a regular expression in a segment for the selected field separated by a forward slash (/). You can revert to the recorded value by using the **Restore Recorded Id** option.

You must be well-versed with the notion of SAP identifier that is used in the SAP GUI scripting so that you can use the regular expression effectively. For more information, refer to the SAP documentation.

SAP server request details

In the test editor, server request elements are located at the end of every SAP screen and provide information that the server returns for the selected screen.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

SAP Screen

Name

This is the name of the current SAP transaction code. This field is read-only.

Program

This is the name of the SAP source program that is currently running. This field is read-only.

Flushes

This is the count of the number of flushes in the automation queue during server communication. This field is read-only.

Response Time

This is the delay between the moment the SAP GUI client sends the request to the SAP server and the moment the server response arrives. The units are milliseconds. This field is read-only.

Interpretation Time

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, not SAP server performance. The units are milliseconds. This field is read-only.

Roundtrips

This is the count of token switches between the SAP GUI client and the SAP server to perform the request. This field is read-only.

Request Time Verification Point

Enable verification point

When selected, the test verifies whether the request time returned by the server is below the specified threshold value. An error is reported in the test log if the measured request time is above the threshold.

Response time threshold (ms)

This is the request time limit above which an error is reported in the test log.

Request Timeout

Timeout value (ms)

Select this option to change the default timeout value (3 minutes) for very long transactions.

Response time threshold (ms)

The test verifies that the request time returned by the server is below the specified threshold value. An error is reported in the test log if the measured request time is above the threshold.

SAP batch connection details

In SAP batch input tests, SAP batch connections contain the basic connection information for a batch input test to connect to the SAP server without a SAP GUI. In most cases, these details are the same as those used when you connect manually to SAP with the SAP GUI.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

SAP Batch Input Connection

Client

This is the SAP client number that is used by the batch input test to connect to the SAP server. You can use data correlation to substitute this value.

User

This is the user name that the batch input test uses to connect to the SAP server. You can use data correlation to substitute this value.

Password

This is the password that the batch input test uses to connect to the SAP server. You can use data correlation to substitute this value.

Language

This is the two-letter language code. You can use data correlation to substitute this value.

Host

This is the IP address or computer name of the SAP server. You can use data correlation to substitute this value.

System Number

This is the system number of the SAP server. You can use data correlation to substitute this value.

Additional SAP Connection Properties

Use this list to specify any advanced SAP Java™ Connector (JCo) properties for advanced SAP router setup. Select the JCo property that you want to set in the **Property name** list, and type the required value in **Property value**. Click **Add** to add more properties.

Test Connection

Use this button to test the connection to the SAP server.

SAP batch input transaction details

In the SAP batch input tests, SAP batch input transactions are located in transactions and are recorded transactions that are to be run at a low level, without a SAP GUI, in order to produce a load on the SAP server.

SAP element label

This is the name of the selected SAP test element as it is displayed in the Test Contents. Use this field to rename the test element, or click **Restore Default** to revert to the default name.

SAP Batch Input Transaction

Code

This is the SAP transaction code of the recorded transaction.

Mode

This is the mode of the batch input transaction as it was recorded in the SAP GUI.

Data table

This is the data table of the batch input transaction as it was recorded in the SAP GUI. See the SAP documentation for details on the contents of the recording.

Working with Selenium or Appium tests

You can manage your Selenium or Appium Java™ tests from the UI Test perspective, where you can import and modify the tests, add them to compound tests, and run them. You can also create new Appium Java™ tests from the Perfecto perspective on Windows™ and Mac OS, add the tests to compound tests, and run them. Perfecto is not currently supported on Linux®.

Supported tests: Java™-based Selenium scripts, such as JUnit 3 and 4, and Java™ main programs.

Selenium tests are run on desktop browsers, while Appium tests are used for web or native applications that run on mobile devices.

Note: You can find more information on the Appium site.

JUnit Appium tests can be run in a Appium framework or in a Perfecto cloud environment.

- To run Appium tests in a Appium framework, the Extension for Selenium/Appium tests must be installed with Rational® Functional TesterHCL OneTest™ UI on the same machine. The Extension for Appium is delivered from version 9.1.0.1 of the product. The Appium server must be previously set up on a machine. You need to reference the server URL in the tests to establish a connection between the workbench and the Appium server. When you run the test from Rational® Functional TesterHCL OneTest™ UI, the requests are sent to the server and executed on the device that is connected to the machine where the server is running. The Appium server can be installed with Rational® Functional TesterHCL OneTest™ UI on the same computer for a local execution of the test.
- To run Appium tests on a cloud Perfecto environment from Rational® Functional TesterHCL OneTest™ UI, the Rational® Functional TesterHCL OneTest™ UI Extension for Perfecto Mobile must be installed on the same machine. This extension is available from version 9.1.1 of the product. The test must reference the URL to the cloud, your credentials, the name and version of the device. When you run the test from the workbench, the requests are sent to the server and executed on the virtual device.

When you use Rational® Functional TesterHCL OneTest™ UI to manage Selenium or Appium tests, you can do the following operations:

- Create new Appium Java™ tests using the Rational® Functional TesterHCL OneTest™ UI Perfecto extension
- Import Java™ projects containing Selenium or Appium tests into the product
- Add Selenium or Appium tests to Test Workbench projects
- Add Selenium or Appium tests into larger workflows containing other tests such as performance tests. Such larger workflows are called compound tests.
- View and modify Selenium and Appium tests
- Run Selenium or Appium tests from the Test Navigator
- Run compound tests that contain Selenium or Appium tests
- View test logs

Importing Selenium or Appium Java tests

You can import Selenium or Appium tests into Rational® Functional TesterHCL OneTest™ UI. The tests are then displayed in the **Test Navigator** from where they can be modified and run. You can also add Selenium or Appium tests to a compound test, which is a combination of different kinds of tests that can be managed using Rational® Functional TesterHCL OneTest™ UI.

From the UI Test perspective, you can:

- Import existing Selenium or Appium Java projects.
- Import a Selenium or Appium test into a **Test Workbench** project.
- Add a Selenium or Appium test to a compound test.

You can view Selenium or Appium tests that you imported in the **Test Navigator**.

Note: If the Selenium or Appium test has a compilation error, the test does not show up in the External Tests folder of the **Test Navigator** view. You must fix the compilation error.

Importing a Selenium or Appium Java project into the UI Test perspective

If you have an existing Java project that contains a Selenium or Appium test, you can import it into the UI Test perspective.

About this task

Complete these steps to import an existing Java project containing a Selenium or Appium test into the UI Test perspective.

1. In the UI Test perspective, click **Import** from the **File** menu.
2. In the Import dialog box, expand **General**, select **Existing Projects into Workspace** and then click **Next**.
3. Click **Browse** and navigate to the root directory where the Java project is located, and then click OK.
   Result
   The projects in the directory you select are listed in the Projects field.
4. Select the check box next to the projects you want to import, and click **Finish**.
   Result

The selected project is displayed in the Test Navigator. In the Logical View, the Selenium test is displayed under the **External Tests** folder.

5. You can now run, view or modify the Selenium or Appium test.

Importing a Selenium or Appium test into a Test Workbench project

To work with an existing Selenium or Appium test, import it into a project in the Web UI Test perspective.

1. In the UI Test perspective, click **Import** from the **File** menu.
2. In the **Import** dialog box, expand **General**, select **File System** and then click **Next**.
3. Specify the directory in which the Selenium or Appium test resides. Click **Browse**.
   Result
   By default, the Selenium or Appium test is imported into the Test Workbench project folder, displayed in the **Into folder** field.
4. The contents of the directory you selected are displayed. Select the components you want to import.
5. To organize the imported test components under a top-level folder under the Test Workbench project, select the **Create top-level folder** check box.
6. Click **Finish**.
   Result
   The imported Selenium or Appium test is displayed under the External Tests folder under the Test Workbench project in the Test Navigator, in the Logical View.

   Note: If some of the tests are not displayed in the External Tests folder, the classpaths may not have been set properly. Ensure that the correct classpaths have been set and that the project compiles successfully. To see the compile errors, ensure that the Show Java Content option in the Test Navigator is enabled.

7. Run, view or modify the imported Selenium or Appium test as required.

Adding a Selenium or Appium test to a compound test

The product provides a one-stop testing environment to work with different types of tests such as Selenium, Appium, Web UI, and functional tests. You can combine such different types of tests to form a single larger workflow, which is called a compound test. When you run a compound test, its constituent tests are run in the defined sequence. You can add Selenium tests to a compound test in the UI Test perspective.

Before you begin
Before adding a Selenium or Appium test to a compound test, ensure that you have imported the test into a **Test Workbench** project in a workspace.

1. In the Test Navigator, double-click the compound test to which you want to add the Selenium or Appium test.
   Result
   The contents of the compound test are shown in the **Compound Test Contents** panel in the Compound Test editor.

2. Select the compound test in the **Compound Test Contents** panel in the Compound Test editor, and do one of the following:
    - Click **Add** to add a Selenium or Appium test as the first element in the compound test.
    - To insert a Selenium or Appium test before a specific element in the compound test, select the element and click **Insert**.

    Result

    The **Select Tests** dialog box is opened, and the tests found in the Eclipse Client workspace are displayed.
3. Select the Selenium or Appium test you want to add to the compound test, and click **OK**.

    Result

    The Selenium or Appium test is added to the compound test, and is displayed as part of the contents of the compound test. When you click the Selenium or Appium test in the compound test element list, its details are displayed in the **Compound Test Element Details** panel in the Compound Test editor.

Viewing and modifying Selenium or Appium tests

You can view and modify Selenium or Appium tests that you have imported into the Web UI Test perspective.

About this task

You can view Selenium or Appium tests in the Logical and Resource Views in the **Test Navigator**. From any of these views, you can open the test in the Java editor and view and modify the script.

Note: If the Selenium or Appium test has a compilation error, the test does not show up in the External Tests folder of the **Test Navigator** view. You must fix the compilation error.

- In the Logical View of the **Test Navigator**, Selenium or Appium tests are listed in the External Tests folder under the project into which they were imported. Double-click the Selenium or Appium test under the External Tests folder to open it in the Java editor.

  Result

  In the **Resource** View, the tests under a project are displayed in the project folder. Double-click the Selenium or Appium test to open it in the Java editor. You can modify the script in the Java editor.
- A Selenium or Appium test that is added to a compound test is displayed the **Compound Test Contents** panel in the Compound Test editor.
    - Click the Selenium or Appium test in the **Compound Test Contents**, to display its details in the **Compound Test Element Details** panel. The name of the test, test path, source type and execution mode are displayed.
    - Click the name of the test in the **Compound Test Element Details** panel to open the test script in the Java editor, in which you can modify the script.

Running Selenium or Appium tests

You can run Selenium or Appium tests that you have imported into a **Test Workbench** project. You can also run a compound test to which you have added a Selenium or Appium test.

About this task

You can run individual Selenium or Appium tests that you have imported into a **Test Workbench** project from the **Test Navigator**.

It is not possible to run Selenium or Appium tests within a compound test individually from the **Compound Test** editor. To run such a Selenium or Appium test individually, run it from the project into which the test has been imported.

1. Do one of the following:
    - To run a Selenium or Appium test that you have imported into a **Test Workbench** project, select the test from the External Tests project folder in the Logical View of the **Test Navigator**.
    - To run an individual Selenium or Appium test that you have added to a compound test, select the Selenium test from the External Tests folder of the project into which the test has been imported.
2. Click the **Run As** icon on the toolbar. The test runs. To run a launch configuration option, click the arrow beside the **Run As** icon and select Run Configuration. Select the desired configuration option and run the test.

Results

The test runs and the run progress is shown in the Console view. When the test run is completed, the test log is displayed.

For the tests executed in a Perfecto environment on page 733, the Perfecto Report window opens automatically onto the **Overview** page where you can see the status of the compound test while it is running. When the test is completed, you need to click Perfecto Reports in the **Overview** page to see the links to the Perfecto reports.

Compound tests

You can create compound tests to help you organize smaller tests into scenarios that can then be run end-to-end. You can combine tests from different extensions to achieve end-to-end flow.

If you need to combine various tests into a single workflow or end-to-end scenario, you can organize the tests into a compound test. Each test may perform a part of the scenario. Each test may also run in a different domain, for example, different web browsers. A typical example of a compound test is an online buying workflow. You may have built smaller tests for each part of an online purchase transaction, such as "log on", "log out", "view item", "add to cart", and "check out". You can combine these tests into a single flow in a compound test. When the compound test is run, its individual tests are run in sequence.

The types of tests you can combine into a compound test depend on the testing capabilities you have purchased. You can also shell-share IBM® Rational® Test WorkbenchHCL OneTest™ Studio family products to add multiple tests into a compound test.

To build the scenario you require in a compound test, you can also add the following annotations:

- Comments
- Synchronization points
- Loops

- Delays
- Transaction folders
- IF-THEN-ELSE
- Tests that are mandatory, using the **Finally** blocks
- Tests to be run in random order, using the **Random Selector**

Creating a compound test

You can create a compound test to organize smaller tests into scenarios that can be run simultaneously or sequentially. You can combine various tests to speed up and minimize the test effort.

Before you begin

You must have created or recorded more than a single test. Refer to the following tasks to create or record a test:

- Web UI tests, see Recording a Web UI test on page 545.
- Android mobile tests, see Recording mobile tests for Android applications on page 692.
- iOS mobile tests, see Recording mobile tests for iOS applications on page 701.
- Sap tests, see Recording an SAP test on page 716.
- Windows test, see Recording a Windows test on page 709.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Perform any of the following actions:
   - If you want to create the compound test as a new project, go to step 3 on page 738.
   - If you want to create the compound test in an existing project, go to step 4 on page 739.
3. Go to **File > New > Compound test**.

   The **New Compound Test** window is displayed.

   You must perform the following steps:

   a. Click the **Create the parent folder** icon.

      The **New Project** pane is displayed.

   b. Enter a name for the project in the **project name** field.

      By default, the test is saved in the workspace location.

      Note: If you want, you can select a different location to save the project.

      Clear the **Use default location** option, and then click **Browse** to select a location to save the test.

   c. Click **Next**, and then select the types of assets you need in your project.
   d. Click **Finish**.

      A new project is created.

e. Enter a name for the test in the **File name** field.
f. Click **Finish**.

Note: The file extension `testsuite` is added to the file name and the compound test is created under **Compound Tests** folder of the new project.

To add the individual tests to the compound tests, go to step 5 on page 739.

4. Right click on the project from the **Test Navigator** pane and perform the following actions.
   a. Go to **New > Compound test**.

      The **New Compound Test** pane is displayed.

   b. Enter a file name for the test, and then click **Finish**.
   c. Click **Finish**.

Note:

The file extension `testsuite` is added to the file name and the compound test is created under **Compound Tests** folder of the selected project.

To add individual tests to **Compound Tests** folder, go to step 5 on page 739.

5. Click **Add** in the **Compound Test** window, and then select **Test**.
   The **Select Tests** window is displayed with all the projects in workspace.
6. Select the tests.
7. Click **OK**.
   The tests are added to the compound test.
8. Go to **File > Save**, to save the compound test.

Results

You have created a compound test which contains multiple tests.

What to do next

You can run the compound test on mobile devices or simulators or emulators, See Configuration of compound test runs.

Viewing compound tests

You can view a compound test in the Compound Test Editor.

About this task

When you open a workspace, the tests and projects that reside in the workspace are listed in the Test Navigator.

You can view compound tests in the Logical and Resource Views in the Test Navigator. From any of these views, you can open the test in the Compound Test Editor.

- In the Logical View of the Test Navigator, compound tests are listed in the Compound Tests folder under the project into which they were imported. Double-click the compound test under the Compound Tests folder to open it in the Compound Test Editor.
  Result
  In the Resource View, all tests under a project are shown in the project folder. Double click the compound test under the project folder to open it in the Compound Test Editor.
- In the Java perspective, compound tests under a project are shown under the root project folder. Double click the compound test under the project folder to open it in the Compound Test Editor.
- The Compound Test Editor contains two panels - the **Compound Test Elements** panel, where the elements of the workflow are listed. Click one of the elements, and its details are displayed int the far right portion of the right panel, which is the **Compound Test Element Details** panel. Double-click any of the test or the test elements to view its details. The name of the test, test path, source type and execution mode are displayed.

Adding tests into a compound test

After creating a compound test, you can add the smaller test pieces that contribute to the larger workflow you are constructing with the compound test. When you run a compound test, each of the tests added to it are invoked in the sequence defined.

You can add many tests of the same type, or different types, to a compound test, depending on the testing requirements.

To add tests to a compound test, complete these steps:

1. In the Test Navigator, double-click the compound test to which you want to add a test. The contents of the compound test are shown in the **Compound Test Contents** panel in the Compound Test editor.
2. Do one of the following:
   - Click **Add** to add a test as the first element in the compound test.
   - To insert a test before a specific element in the compound test, select the element and click **Insert**.

   The **Select Tests** dialog box is opened, and the tests found in the Eclipse Client workspace are displayed.
3. Select the test you want to add to the compound test, and click **OK**. The test is added to the compound test, and is displayed as part of the elements of the compound test in the **Compound Test Contents** panel. When you click the test you added, its details are displayed in the **Compound Test Element Details** panel in the Compound Test editor.
4. Save your changes.

In addition to the tests that you can add to a compound test, you can also add the following elements to construct the workflow you need:

- Comments to document the test
- Delays in the test

- Synchronization points
- Loops
- Transaction folders
- Parts of the test that are mandatory
- Tests to be run in random order

Modifying a compound test

You can modify a compound test in the Compound Test Editor.

About this task

A compound test is a testing workflow comprising smaller tests and other test elements in a certain sequence. You might want to order the tests and test elements to suit your workflow requirement, or add further tests and elements.

1. In the Test Navigator, double-click the compound test that you want to modify. Its elements are shown in the **Compound Test Contents** right panel in the Eclipse Client.
2. To add a test or test element at the beginning of the compound test elements list, select the compound test in the **Compound Test Contents** panel, click **Add**, and then click **Test**. To insert a test or test element into the test, select the test element before which the insertion must be made, and click **Insert**.
3. Add or insert the test or test element you need, and click **OK**. The modified compound test displays its updated elements in the **Compound Test Contents** right panel.
4. Save your changes.

Running compound tests

When you run a compound test, its test elements are run in the order defined in the compound test.

About this task

When you run a compound test, you are prompted to open the Test Execution perspective, in which details of the test run are displayed. When the test run is complete, the Test Log displays the run results. For details about running a compound test that contains Web UI tests on multiple browsers simultaneously, see Running a Web UI test on page 1243.

Prior to 9.2, text execution would terminate on a fatal exception in any of the tests in a compound test. Starting from 9.2, there is a new preference to allow text execution for a compound test to continue after a fatal exception in one of the tests. To set the preference, see **Window > Preferences > Test > Test Execution > Error handling > UI Fatal Error**.

1. In the Test Navigator, select the compound test to run.
2. Click the Run As icon on the toolbar. The test runs. To run a launch configuration option, click the arrow beside the Run As icon and select Run Configuration. Select a configuration option and run the test.
   Result
   The **Confirm Perspective Switch** dialog box is opened, prompting you to switch to the Test Execution perspective. Click **Yes**.
3. Select an option to run the test.

Result

The Test Execution perspective is opened and the test runs. On completion, the test log is displayed.

Results

You can work with the test log by exporting it into a flat file.

Generating compound test result reports

When a compound test run is completed, a Test Log is shown in the Test Execution perspective. You can work with the information in the test log and also generate test result reports.

Exporting the Test Log

When a compound test run is completed, a Test Log is displayed in the Test Execution perspective.

About this task

The Test Log displays the following details:

- The General Information tab displays the name of the compound test and its description. The location of the test log file is also shown.
- The Common Properties tab shows the verdict of the test results.
- The Verdict Summary and Verdict List tabs provide a pie chart of verdicts for different components of the test, and a list of the first 20 verdicts. You can view details about the verdicts by clicking the links in the Verdict List tab.

You can export the contents of the test log to a full-text file.

1. To export the contents of the test log to a full-text file, right-click the test run result under the Results folder of the compound test, and click **Export Test Log**.
2. In the **Export Test Log** dialog box, specify where the test log should be exported to, in the **Location** field.
3. Select the format in which the log must be exported, from the list in the **Export Format** field. You can select either Flat Text - Default Encoding or Flat Text - Unicode Encoding.
4. Click **Finish**.

   Result

   The test log is exported as a full-text file, with the test results run name, to the location you specified.

Generating a functional test report

You can generate a functional test report from the test run results as a HTML file.

About this task

When you generate a functional test report as a HTML file, the following details are displayed in the report:

- A global summary, which lists the number of tests run, verification points, defects
- A test summary which displays the name of each test, the start and end times and the verdicts.

1. Test run results are displayed under the Results folder of a project. Right-click the test run result you want to view and click **Generate Functional Test Report**.
   Result
   The **Generate Functional Test Report** dialog box is opened.
2. Select the parent folder in which the report must be stored.
3. By default, the name of the compound test and the date and time stamp is displayed as the name of the report in the **Name** field. You can change the name.
4. Click **Next**.
5. Select the report template to be used. If you select the Common Functional Test Report (XSL) format, the report is generated as a HTML file. If you select the Common Test Functional Report format, you can select either the HTML or PDF output format.
6. Click **Finish**.
   Result
   The report is generated and displayed. The report is listed under the Functional Reports folder under the compound test in the Test Navigator.

Creating an executive summary

You can create an executive summary or test statistics report from the test run results. Executive summaries are generated according to the type of test.

About this task
An executive summary displays the tests and methods that were run, and their success or failure information. This information is shown in summary charts as well as in bar graphs.

1. Under the Results folder of the project, right-click the test run result you want to view and click **Create Executive Summary**.
   Result
   The **Generate Functional Test Report** dialog box is opened.
2. Select the type of test report you want to generate.
3. Click **Finish**.
   Result
   The report is generated and displayed. The report is listed under the Functional Reports folder under the compound test in the Test Navigator.

Adding a compound test to a Test Workbench project

You can create a compound test in a test workbench project. If you have an existing compound test, you can import the test to a test workbench project.

Creating a compound test in a test workbench project

You can create a compound test in a test workbench project.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Perform any of the following actions:
   - If you want to create the compound test as a new project, go to step 3 on page 744.
   - If you want to create the compound test in an existing project, go to step 4 on page 744.
3. Go to **File > New > Compound test**.

   The **New Compound Test** window is displayed.

   You must perform the following steps:

   a. Click the **Create the parent folder** icon.

      The **New Project** pane is displayed.

   b. Enter a name for the project in the **project name** field.

      By default, the test is saved in the workspace location.

      Note: If you want, you can select a different location to save the project.

      Clear the **Use default location** option, and then click **Browse** to select a location to save the test.

   c. Click **Next**, and then select the types of assets you need in your project.
   d. Click **Finish**.

      A new project is created.

   e. Enter a name for the test in the **File name** field.
   f. Click **Finish**.

      Note: The file extension `testsuite` is added to the file name and the compound test is created under **Compound Tests** folder of the new project.

   To add the individual tests to the compound tests, go to step 5 on page 745.
4. Right click on the project from the **Test Navigator** pane and perform the following actions.
   a. Go to **New > Compound test**.

      The **New Compound Test** pane is displayed.

   b. Enter a file name for the test, and then click **Finish**.
   c. Click **Finish**.

Note:

The file extension `testsuite` is added to the file name and the compound test is created under **Compound Tests** folder of the selected project.

To add individual tests to **Compound Tests** folder, go to step 5 on page 745.

5. Click **Add** in the **Compound Test** window, and then select **Test**.
   The **Select Tests** window is displayed with all the projects in workspace.
6. Select the tests.
7. Click **OK**.
   The tests are added to the compound test.
8. Go to **File > Save**, to save the compound test.

Importing a compound test into a Test Workbench project

You can import a compound test into a test workbench project.

1. In the Web UI Test perspective, in the Test Navigator, right-click the test workbench project into which you want to import the compound test and click **Import**.
2. In the **Import** dialog box, expand **General** in the source list, select **Import test assets with dependencies** and then click **Next**.
3. Specify the directory in which the compound test resides. Click **Browse**.
   Result
   By default, the compound test is imported into the test workbench project folder.
4. The compound test assets in the folder you selected are displayed. Select the components you want to import.
5. Click **Finish**.
   Result
   The imported compound test is displayed in the **Compound Test Elements** panel in the Compound Test editor.

Accelerated Functional Tests

When you create Web UI, mobile, or Windows tests, you can use the Accelerated Functional Test (AFT) feature in Rational® Functional TesterHCL OneTest™ UI to distribute test effort during the play back of the recorded tests.

You can find the following information about creating AFT Suites:

- Creating an Accelerated Functional Test asset on page 746
- Creating an AFT Suite to run Web UI tests on BrowserStack Cloud on page 747
- Creating an AFT suite for mobile tests on page 749
- Creating an AFT Suite to run mobile tests on the BitBar cloud on page 754
- Creating an AFT Suite to run mobile tests on BrowserStack Cloud on page 757
- Creating an AFT Suite to run mobile tests on the pCloudy cloud on page 760

- Creating an AFT Suite to run mobile tests on the Perfecto cloud on page 763
- Creating an AFT Suite to run the tests on multiple mobile clouds on page 767

Creating an Accelerated Functional Test asset

You can create an Accelerated Functional Test asset while running tests by using the distributed test option. When you create an Accelerated Functional Test asset, an XML file is generated that contains the selected tests along with the browsers. You can also add more tests, browsers, devices, and agents to extend the tests.

Before you begin

You must have created one or more Web UI tests or compound tests.

About this task

Accelerated functional testing supports two types of parallel execution. The executions are as follows:

- Execution of a single Web UI or compound test on multiple browsers: This execution is supported on all installed browsers such as Microsoft Internet Explorer, Microsoft Edge, Google Chrome, Mozilla Firefox, and Apple Safari simultaneously. This combination can be specified in the XML file to execute the tests locally as well as remotely. If you want to execute the tests remotely, you must specify the remote agents in the XML file.
- Execution of multiple tests on multiple browsers: This execution is supported only on Mozilla Firefox and Google Chrome browsers. It is also supported on Chrome device mode.

1. From your test navigator, select and right-click the required Web UI or compound tests for which you want to run the distributed tests.
2. Right-click the selected tests and click **Run Distributed Tests**. The **Run Accelerated Functional Test** dialog box containing the selected tests is displayed.

   Note: This is the only method to run an AFT suite.

3. Click **Add** to add more tests to the distributed test. Click **Remove** to remove the required tests.
4. Click **Save as** to rerun the same set of tests that you have selected. The tests are saved as an accelerated functional test asset and generate an XML file that you can use to run the distributed tests later.

   Note: If you do not select the **Save as** check box, then the tests are not saved as an accelerated functional asset and the distributed tests run only once.

5. Click **Browse** to select a location to save the test asset and provide a name for the test asset.
6. Click **Next**.
7. In the **Select browser(s)** dialog box, select the required browser and click **Finish**.

   The accelerated functional test asset starts running the XML file is now created. To execute the accelerated functional test asset by using the XML file, complete the following steps:

a. From the test navigator, right-click the accelerated functional asset XML file and click **Run Distributed Tests**. The **Run Accelerated Functional Test** dialog box is displayed.You can choose to select the following checkboxes:
   - **Re-run failed tests only from last playback**: Select this checkbox if you want to rerun only the failed tests from the previous playback.

     Note: If this option is enabled, the failed tests are rerun only on the browsers and location on which the test failed previously.

   - **Fix the browser-driver incompatibility**: Select this checkbox to automatically resolve the incompatibility between the browser and the driver, while you play back the AFT tests.

     Tip: As the playback starts only after the appropriate driver is downloaded, a timeout error might occur if the application is not started within the time limit specified in the **Time Out** field. You must increase the time in the **Time Out** field. To resolve this error, you can modify the timeout value. The default timeout is 10 seconds. To modify the timeout, check the option and enter a new value.

b. Click **OK**.

Results

You have created an Accelerated Functional Test asset.

Related information

Unable to run a distributed test with the Save as option

Configuration of test runs from the command line on page 1381

Creating an AFT Suite to run Web UI tests on BrowserStack Cloud

You can create an AFT Suite when you want to run multiple Web UI tests on wide range of devices that are available on the BrowserStack Cloud. You can configure the AFT Suite with details of the tests, devices to use, and the location from where the tests must be initiated.

Before you begin

You must have completed the following tasks:

- Created or recorded Web UI tests for the web application that you want to test.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Noted the IP address of the remote agent computer if the device cloud is connected to the remote computer.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.

3. Perform the following actions in the **Select a wizard** dialog:
   a. Select the **XML File** option in the **XML** folder.
   b. Click **Next**.

      The projects that contain the tests are displayed.

   c. Select the project where you want to save the XML file.
   d. Enter a name for the XML file in the **File name** field.
   e. Click **Next**.
   f. Select the **Create XML file from an XML template** option.
   g. Click **Finish**.

      An XML document is displayed for editing in the **Source** view tab.

      Note: Open the **Source** view tab if the **Design** view tab is displayed.

      A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_Web_UI_test_1>"/>
     <test path="<path_to_Web_UI_test_2>"/>
  </tests>
  <browsers>
     <browser name="<name_of_the_browser>" id="<device_cloud_name> : <device_name>"/>
   </browsers>
  <locations>
     <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Sample value	Description
inits	*id*	*<any_alphanumeric_value>*	N2D6E3E008FC31EDAE07D38A0D9588E9	Specifies any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_Web_UI_test_1><path_to_*	/UI1/Tests/webap-p1.testsuite	Specifies the path to the Web UI test in your project.

Element	Attribute	Variable	Sample value	Description
		Web_UI_test_-2><path_to_-Web_UI_test_-3>	/UI1/Tests/webap-p2.testsuite /UI1/Tests/webap-p3.testsuite	**Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
browser	*name*	*<name_of_-the_browser>*	Chrome	Specifies the name of the browser to be used to run the test on the device cloud.
	id	*<device_-cloud_-name> : <de-vice_name>*	BrowserS-tack : Google Pixel 7 Pro	Specifies the name of the device cloud and ID of the device on the device cloud.
location	*host*	*<location_-where_test_-is_run>*	localhost	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional Tester-HCL OneTest™ UI or the remote agent com-puter.

4. Enter the values for the attributes for your tests, device, and location in the XML file.
5. Save the XML file.

Results

You have created an AFT Suite for Web UI tests, which you can use to run the tests on devices that are on the BrowserStack Cloud.

What to do next

You can initiate a run of the Web UI tests that are in the AFT Suite. See Running Web UI tests as an AFT Suite on BrowserStack Cloud on page 1365.

Creating an AFT suite for mobile tests

When you want to run a single mobile test on multiple devices, multiple mobile tests on a single device, or multiple mobile tests on multiple devices that are hosted on a local computer or remote agent computer, you can create an Accelerated Functional Test suite (AFT) suite for mobile tests.

Before you begin

You must have completed the following tasks:

- Connected the devices through a USB cable and configured the emulators or simulators on the computer that runs Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.
- Been aware of the following details:
    - The IP address of the UI Test Agent.
    - The port number of the UI Test Agent.



Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

About this task

To run the tests as an AFT suite, you must create an XML file where you can specify the test details. You can use this XML file to run these tests anytime later and you can also add more tests, and devices to extend the tests.

You can refer to the following table to know the format of the device id for each of the devices that you use in the AFT suite:

Device name	Device id format
Android emulator	**Emulator:<Name of Android emulator with the space replaced with underscore>** For example, if the emulator name is **Pixel 2 API 30**, then device id is **Emulator:Pixel_2_API_30**
Android real device	**Android:<Name of Android real device with the space replaced with underscore>** For example, if the real device name is **Pixel 4**, then device is **Android:Pixel_4**
iOS simulator	**Simulator:<Name of iOS simulator_iOS version>** For example, if the iOS simulator name is **iPhone 11 Pro** and iOS version is **14.4**, then device id is **Simulator:iPhone 11 Pro_14.4**
iOS real device	**iOS:<Name of iOS device_iOS version_UUID>** For example, if the iOS real device name is **My iPhone**, iOS version is **14.4**, and device UUID is **445f47e79c803c95cd8ef4f2429c61e0b032abdc**, then device id is **iOS:My iPhone_14.4_445f47e79c803c95cd8ef4f2429c61e0b032abdc**

Restriction:

- The AFT suite must contain either Android tests or iOS tests.
- The AFT suite must contain only mobile tests or a compound test that contains only mobile tests.
- You cannot run one or more single mobile tests and compound tests that contain a mix of mobile and Web UI or any other supported tests in an AFT suite.
- Each group can contain only one location. You can use multiple groups to run tests on multiple locations. You cannot have multiple groups for the same location.
- Each group can contain either mobile test scripts only or a compound test that contains mobile test scripts.

1. Create an XML file to specify the details of the test suites, devices, and the location by performing the following steps:
   a. Click **File > New > Other**.

      The **Select a wizard** dialog box is displayed.

   b. Select the **XML File** in the **XML** section and then click **Next**.

      The available projects are displayed.

   c. Select a project where you want to save the XML file, enter a name for the XML file in the **File name** field, and then click **Next**.
   d. Select the **Create an XML file from an XML template** option and click **Finish**.

      A blank XML document opens.

   e. Click the **Source** tab of the XML document.
   f. Provide the necessary details in the XML file, and then save the file.

      A sample format of the XML file is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="N3E78C8006B211EB9003B7CBBD8B707A"
type="com.ibm.rational.test.ft.aftsuite">
<group>
    <tests>
        <test path="<Path of the mobile test1>" />
        <test path="<Path of the mobile test2>" />
    </tests>
    <devices>
        <device id="<Device type1:Device id1>" />
        <device id="<Device type2:Device id2>" />
    </devices>
    <locations>
        <location host = "<local or remote host>"
        appium.server.host = "<IP address of the UI Test Agent>"
        appium.server.port = "<Port number of the UI Test Agent>" />
    </locations>
</group>
```

```
<group>
    <tests>
        <test path="<Path of the mobile test3>" />
        <test path="<Path of the mobile test4>" />
    </tests>
    <devices>
        <device id="<Device type1:Device id3>" />
        <device id="<Device type2:Device id4>" />
    </devices>
    <locations>
        <location host = "<local or remote host>"
        appium.server.host = "<IP address of the UI Test Agent>"
        appium.server.port = "<Port number of the UI Test Agent>" />
    </locations>
</group>
</inits>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="N3E78C8006B211EB9003B7CBBD8B707A"
type="com.ibm.rational.test.ft.aftsuite">
<group>
    <tests>
        <test path="<Path of the mobile test1>" />
        <test path="<Path of the mobile test2>" />
    </tests>
    <devices>
        <device id="<Device type1:Device id1>" />
        <device id="<Device type2:Device id2>" />
    </devices>
    <locations>
        <location host = "<local or remote host>"
        appium.server.host = "<IP address of the UI Test Agent>"
        appium.server.port = "<Port number on which UI Test Agent is running>"/>
    </locations>
</group>
<group>
    <tests>
        <test path="<Path of the mobile test3>" />
        <test path="<Path of the mobile test4>" />
    </tests>
    <devices>
        <device id="<Device type1:Device id3>" />
        <device id="<Device type2:Device id4>" />
    </devices>
    <locations>
        <location host = "<local or remote host>"
        appium.server.host = "<IP address of the UI Test Agent>"
        appium.server.port = "<Port number on which UI Test Agent is running>"/>
    </locations>
</group>
</inits>
```

In the following example, you can see that for each *group* of tests, you can specify a single location only.

For example,

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="N04408500ADA11EABD15B2C0CB4A9F45"
 type="com.ibm.rational.test.ft.aftsuite">
```

```
<group>
    <tests>
        <test path="/MobileProject1/CmpTest1.testsuite">
        </test>
    </tests>
    <devices>
        <device id="Emulator:Pixel_API_29"></device>
        <device id="Emulator:Pixel_API_30"></device>
    </devices>
    <locations>
        <location host="localhost" appium.server.host="localhost"></location>
    </locations>
</group>
<group>
    <tests>
        <test path="/MobileProject1/Tests/Copy of APIDemoDebug_Accessibility1.testsuite"></test>
        <test path="/MobileProject1/Tests/Copy of APIDemoDebug_App.testsuite"/>
    </tests>
    <devices>
    <device id="Emulator:Pixel_3a_API_29"></device>
    </devices>
    <locations>
        <location host="10.115.160.202" appium.server.host="10.115.160.202"
 appium.server.port="7082"
       ></location>
    </locations>
</group>
</inits>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="N04408500ADA11EABD15B2C0CB4A9F45"
 type="com.ibm.rational.test.ft.aftsuite">
<group>
    <tests>
        <test path="/MobileProject1/CmpTest1.testsuite">
        </test>
    </tests>
    <devices>
        <device id="Emulator:Pixel_API_29"></device>
        <device id="Emulator:Pixel_API_30"></device>
    </devices>
    <locations>
        <location host="localhost" appium.server.host="localhost"></location>
    </locations>
</group>
<group>
    <tests>
        <test path="/MobileProject1/Tests/Copy of APIDemoDebug_Accessibility1.testsuite"></test>
        <test path="/MobileProject1/Tests/Copy of APIDemoDebug_App.testsuite"/>
    </tests>
    <devices>
    <device id="Emulator:Pixel_3a_API_29"></device>
    </devices>
    <locations>
        <location host="10.115.160.202" appium.server.host="10.115.160.202"
 appium.server.port="7082"></location>
    </locations>
```

```
</group>
</inits>
```

2. Click **OK**.

Results

You have created an AFT suite to run mobile tests.

Related information

Running mobile tests as an AFT suite on page 1339

Creating an AFT Suite to run mobile tests on the BitBar cloud

When you want to run multiple mobile tests recorded on a mobile application, you can create an AFT. You can configure the AFT Suite with details of the tests, devices to use, and the location from where the tests must be initiated. You can then run the mobile tests on devices that are connected to the BitBar Cloud.

Before you begin

You must have completed the following tasks:

- Created or recorded mobile tests for the mobile application that you want to test.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Noted the IDs or names of the devices connected to the BitBar Cloud that you want to use.
- Noted the IP address of the remote agent computer if the mobile cloud is connected to the remote computer.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.

3. Perform the following actions in the **Select a wizard** dialog:
   a. Select the **XML File** option in the **XML** folder.
   b. Click **Next**.

      The projects that contain the tests are displayed.

   c. Select the project where you want to save the XML file.
   d. Enter a name for the XML file in the **File name** field.
   e. Click **Next**.
   f. Select the **Create XML file from an XML template** option.
   g. Click **Finish**.

      An XML document is displayed for editing in the **Source** view tab.

Note: Open the **Source** view tab if the **Design** view tab is displayed.

A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_mobile_test_1>"/>
     <test path="<path_to_mobile_test_2>"/>
  </tests>
  <devices>
     <device id="<name of the device 1>”/>
  </devices>
  <locations>
     <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Description
inits	*id*	*<any_alphanumeric_value>*	Specifies that you can enter any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_mobile_test_1><path_to_mobile_test_2><path_to_mobile_test_3>*	Specifies the path to the mobile test in your project. **Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
device	*id*	*<name_of_the_device_1>*	Specifies the name or id of the device that is connected to the device cloud.
location	*host*	*<location_where_test_is_run>*	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.

4. Create the following elements in the XML document:

Note: You can copy the sample syntax, and then edit the values in the UI. Alternatively, you can create the elements in the XML in the UI.

a. `<group>`

The start and end tags are automatically inserted.

b. Create `<tests>` within the `<group>` tag.
c. Create `<test>` within the `<tests>` tag.
d. Add the attribute path to the `<test>` as follows:

```
<test path="">
```

Note: You must enclose all values that you enter for the attributes within a double quotation mark.

e. Add additional rows for each mobile test.
f. Create `<devices>` within the `<group>` tag.
g. Create `<device>` within the `<devices>` tag.
h. Add the attribute id as follows:

```
<device id=""></device>
```

i. Create `<locations>` within the `<group>` tag.
j. Create `<location>` within the `<locations>` tag.
k. Add the attribute host as follows:

```
<location host=""></location>
```

5. Refer to the following table for the values of the attributes that you can use in the XML file:

Element	Attribute	Example of a value	Explanation
test	*path*	*MyProject\test_1*	The path in the workspace that contains the project with the test. You can find the path of the test from the **Test Navigator** by right-clicking the test, and then clicking **Properties > Resource**.
device	*id*	*Bitbar:Google Pixel 2*	The name of the Android device is Google Pixel 2 that is connected to the BitBar Cloud.
		Bitbar:Apple iPhone 6s Plus A1687 12.0.1	The name of the iOS device is Bitbar : Apple iPhone 6s Plus A1687 12.0.1 that is connected to the BitBar Cloud.

Element	Attribute	Example of a value	Explanation
location	*host*	*localhost*	The host is localhost if the device cloud is connected to the computer that runs Rational® Functional TesterHCL OneTest™ UI.
		1.20.30.40	The host is the IP address of the remote agent computer, if the device cloud is connected to the remote agent computer.

6. Enter the values for the attributes for your tests, device, and location in the XML file.
7. Save the XML file.

Results

You have created an AFT Suite for mobile tests, which you can use to run the mobile tests on devices that are connected to the BitBar cloud.

What to do next

You can initiate a run of the mobile tests that are in the AFT Suite. See Running mobile tests as an AFT Suite on BitBar Cloud on page 1340.

Creating an AFT Suite to run mobile tests on BrowserStack Cloud

You can create an AFT Suite when you want to run multiple mobile tests on wide range of devices that are available on the BrowserStack Cloud. You can configure the AFT Suite with details of the tests, devices to use, and the location from where the tests must be initiated.

Before you begin

You must have completed the following tasks:

- Created or recorded mobile tests for the mobile application that you want to test.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Noted the IDs or names of the devices connected to the BrowserStack cloud that you want to use.
- Noted the IP address of the remote agent computer if the mobile cloud is connected to the remote computer.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.
3. Perform the following actions in the **Select a wizard** dialog:

a. Select the **XML File** option in the **XML** folder.
b. Click **Next**.

   The projects that contain the tests are displayed.

c. Select the project where you want to save the XML file.
d. Enter a name for the XML file in the **File name** field.
e. Click **Next**.
f. Select the **Create XML file from an XML template** option.
g. Click **Finish**.

   An XML document is displayed for editing in the **Source** view tab.

   Note: Open the **Source** view tab if the **Design** view tab is displayed.

   A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_mobile_test_1>"/>
     <test path="<path_to_mobile_test_2>"/>
  </tests>
  <devices>
     <device id="<name of the device 1>”/>
  </devices>
  <locations>
     <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Description
inits	*id*	*<any_alphanumeric_value>*	Specifies that you can enter any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_mobile_test_1><path_to_mobile_test_2><path_to_mobile_test_3>*	Specifies the path to the mobile test in your project. **Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
device	*id*	*<name_of_the_device_1>*	Specifies the name or id of the device that is connected to the device cloud.

Element	Attribute	Variable	Description
location	*host*	*<location_-where_test_is_-run>*	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.

4. Create the following elements in the XML document:

   Note: You can copy the sample syntax, and then edit the values in the UI. Alternatively, you can create the elements in the XML in the UI.

   a. `<group>`

      The start and end tags are automatically inserted.

   b. Create `<tests>` within the `<group>` tag.
   c. Create `<test>` within the `<tests>` tag.
   d. Add the attribute path to the `<test>` as follows:

      ```
      <test path="">
      ```

      Note: You must enclose all values that you enter for the attributes within a double quotation mark.

   e. Add additional rows for each mobile test.
   f. Create `<devices>` within the `<group>` tag.
   g. Create `<device>` within the `<devices>` tag.
   h. Add the attribute id as follows:

      ```
      <device id=""></device>
      ```

   i. Create `<locations>` within the `<group>` tag.
   j. Create `<location>` within the `<locations>` tag.
   k. Add the attribute host as follows:

      ```
      <location host=""></location>
      ```

5. Refer to the following table for the values of the attributes that you can use in the XML file:

Element	Attribute	Example of a value	Explanation
test	*path*	*MyProject\UI1\Tests\api-*	The path in the workspace that contains the project with the test. You can find the path of the test from the **Test Navigator** by right-clicking the test, and then go to **Properties > Resource**.

Element	Attribute	Example of a value	Explanation
		demos1.test-suite	
device	*id*	*BrowserStack:8D3E35CF16D8E3E4B27AB-BCD0E582E2761CA-DA*	The name of the Android device is 8D3E35CF16D8E3E4B27AB827E4827ABBCD0E582E2761CA-DA that is connected to the BrowserStack Cloud.
		BrowserStack:R48904T-NSAZ	The name of the iOS device is BrowserStack : R48904TNSAZ that is connected to the BrowserStack Cloud.
location	*host*	*localhost*	The host is localhost if the device cloud is connected to the computer that runs Rational® Functional TesterHCL OneTest™ UI.
		1.20.30.40	The host is the IP address of the remote agent computer, if the device cloud is connected to the remote agent computer.

6. Enter the values for the attributes for your tests, device, and location in the XML file.
7. Save the XML file.

Results

You have created an AFT Suite for mobile tests, which you can use to run the mobile tests on devices that are on the BrowserStack Cloud.

What to do next

You can initiate a run of the mobile tests that are in the AFT Suite. See Running mobile tests as an AFT Suite on BrowserStack Cloud on page 1342.

Creating an AFT Suite to run mobile tests on the pCloudy cloud

When you want to run multiple mobile tests recorded on a mobile application, you can create an AFT. You can configure the AFT Suite with details of the tests, devices to use, and the location from where the tests must be initiated. You can then run the mobile tests on devices that are connected to the pCloudy Cloud.

Before you begin

You must have completed the following tasks:

- Created or recorded mobile tests for the mobile application that you want to test.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.

- Noted the IDs or names of the devices connected to the BitBar Cloud that you want to use.
- Noted the IP address of the remote agent computer if the mobile cloud is connected to the remote computer.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.

3. Perform the following actions in the **Select a wizard** dialog:
   a. Select the **XML File** option in the **XML** folder.
   b. Click **Next**.

      The projects that contain the tests are displayed.

   c. Select the project where you want to save the XML file.
   d. Enter a name for the XML file in the **File name** field.
   e. Click **Next**.
   f. Select the **Create XML file from an XML template** option.
   g. Click **Finish**.

      An XML document is displayed for editing in the **Source** view tab.

      Note: Open the **Source** view tab if the **Design** view tab is displayed.

      A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_mobile_test_1>"/>
     <test path="<path_to_mobile_test_2>"/>
  </tests>
  <devices>
     <device id="<name of the device 1>”/>
  </devices>
  <locations>
     <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Description
inits	*id*	*<any_alphanu-meric_value>*	Specifies that you can enter any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_-mobile_test_-*	Specifies the path to the mobile test in your project.

Element	Attribute	Variable	Description
		1><path_to_-mobile_test_-2><path_to_mo-bile_test_3>	**Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
device	*id*	*<name_of_the_-device_1>*	Specifies the name or id of the device that is con-nected to the device cloud.
location	*host*	*<location_-where_test_is_-run>*	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.

4. Create the following elements in the XML document:

   Note: You can copy the sample syntax, and then edit the values in the UI. Alternatively, you can create the elements in the XML in the UI.

   a. `<group>`

      The start and end tags are automatically inserted.

   b. Create `<tests>` within the `<group>` tag.
   c. Create `<test>` within the `<tests>` tag.
   d. Add the attribute path to the `<test>` as follows:

      ```
      <test path="">
      ```

      Note: You must enclose all values that you enter for the attributes within a double quotation mark.

   e. Add additional rows for each mobile test.
   f. Create `<devices>` within the `<group>` tag.
   g. Create `<device>` within the `<devices>` tag.
   h. Add the attribute id as follows:

      ```
      <device id=""></device>
      ```

   i. Create `<locations>` within the `<group>` tag.
   j. Create `<location>` within the `<locations>` tag.
   k. Add the attribute host as follows:

      ```
      <location host=""></location>
      ```

5. Refer to the following table for the values of the attributes that you can use in the XML file:

Element	Attribute	Example of a value	Explanation
test	*path*	*MyProject\test_1*	The path in the workspace that contains the project with the test. You can find the path of the test from the **Test Navigator** by right-clicking the test, and then go to **Properties > Resource**.
device	*id*	*pcloudy:Pixel_2_API_30*	The name of the Android device is Pixel 2 API 30 that is connected to the pCloudy Cloud.
		pCloudy:Apple_iPhone8plus_Ios_12.2.0_9db66	The name of the iOS device is Apple_iPhone8plus_Ios_12.2.0_9db66 that is connected to the pCloudy Cloud.
location	*host*	*localhost*	The host is localhost if the device cloud is connected to the computer that runs Rational® Functional TesterHCL OneTest™ UI.
		1.20.30.40	The host is the IP address of the remote agent computer, if the device cloud is connected to the remote agent computer.

6. Enter the values for the attributes for your tests, device, and location in the XML file.
7. Save the XML file.

Results

You have created an AFT Suite for mobile tests, which you can use to run the mobile tests on devices that are connected to the pCloudy cloud.

What to do next

You can initiate a run of the mobile tests that are in the AFT Suite. See Running mobile tests as an AFT Suite on pCloudy Cloud on page 1344.

Creating an AFT Suite to run mobile tests on the Perfecto cloud

When you want to run multiple mobile tests recorded on a mobile application, you can create an AFT. You can configure the AFT Suite with details of the tests, devices to use, and the location from where the tests must be initiated. You can then run the mobile tests on devices that are connected to the Perfecto Cloud.

Before you begin

You must have completed the following tasks:

- Created or recorded mobile tests for the mobile application that you want to test.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.
- Noted the IDs or names of the devices connected to the BitBar Cloud that you want to use.
- Noted the IP address of the remote agent computer if the mobile cloud is connected to the remote computer.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.
3. Perform the following actions in the **Select a wizard** dialog:
   a. Select the **XML File** option in the **XML** folder.
   b. Click **Next**.

      The projects that contain the tests are displayed.
   c. Select the project where you want to save the XML file.
   d. Enter a name for the XML file in the **File name** field.
   e. Click **Next**.
   f. Select the **Create XML file from an XML template** option.
   g. Click **Finish**.

      An XML document is displayed for editing in the **Source** view tab.

      Note: Open the **Source** view tab if the **Design** view tab is displayed.

      A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_mobile_test_1>"/>
     <test path="<path_to_mobile_test_2>"/>
  </tests>
  <devices>
     <device id="<name of the device 1>”/>
  </devices>
  <locations>
     <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Description
inits	*id*	*<any_alphanumeric_value>*	Specifies that you can enter any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_mobile_test_1><path_to_mobile_test_2><path_to_mobile_test_3>*	Specifies the path to the mobile test in your project. **Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
device	*id*	*<name_of_the_device_1>*	Specifies the name or id of the device that is connected to the device cloud.
location	*host*	*<location_where_test_is_run>*	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.

4. Create the following elements in the XML document:

Note: You can copy the sample syntax, and then edit the values in the UI. Alternatively, you can create the elements in the XML in the UI.

a. `<group>`

The start and end tags are automatically inserted.

b. Create `<tests>` within the `<group>` tag.
c. Create `<test>` within the `<tests>` tag.
d. Add the attribute path to the `<test>` as follows:

```
<test path="">
```

Note: You must enclose all values that you enter for the attributes within a double quotation mark.

e. Add additional rows for each mobile test.
f. Create `<devices>` within the `<group>` tag.
g. Create `<device>` within the `<devices>` tag.
h. Add the attribute id as follows:

```
<device id=""></device>
```

i. Create `<locations>` within the `<group>` tag.

j. Create `<location>` within the `<locations>` tag.
k. Add the attribute host as follows:

```
<location host=""></location>
```

5. Refer to the following table for the values of the attributes that you can use in the XML file:

Element	Attribute	Example of a value	Explanation
test	*path*	*MyProject\test_1*	The path in the workspace that contains the project with the test. You can find the path of the test from the **Test Navigator** by right-clicking the test, and then go to **Properties > Resource**.
device	*id*	*Perfecto:8D3E35CF16D8BD827E4827ABBCD0E582E2761CA-DA*	The name of the Android device is 8D827E4827ABBCD0E582E2761CA-DA that is connected to the Perfecto Cloud.
		Perfecto:R48904T-NSAZ	The name of the iOS device is Perfecto : R48904T-NSAZ that is connected to the Perfecto Cloud.
location	*host*	*localhost*	The host is localhost if the device cloud is connected to the computer that runs Rational® Functional TesterHCL OneTest™ UI.
		1.20.30.40	The host is the IP address of the remote agent computer, if the device cloud is connected to the remote agent computer.

6. Enter the values for the attributes for your tests, device, and location in the XML file.
7. Save the XML file.

Results

You have created an AFT Suite for mobile tests, which you can use to run the mobile tests on devices that are connected to the Perfecto cloud.

What to do next

You can run the AFT suite on mobile devices connected to Perfecto mobile device clouds. See Running mobile tests as an AFT Suite on Perfecto Cloud on page 1345.

Creating an AFT Suite to run the tests on multiple mobile clouds

When you want to run the mobile tests simultaneously on devices connected to multiple clouds, you can create an AFT. You can configure the AFT Suite with details of the tests, devices on multiple mobile clouds, and the location from where the tests must be initiated.

Before you begin

You must have completed the following tasks:

- Created or recorded mobile tests for the mobile application that you want to test.
- You must have set up accounts and obtained valid credentials for more than one mobile cloud.
- Noted the IDs or names of the devices connected to the mobile clouds that you want to use.
- Noted the IP address of the remote agent computer if the mobile cloud is connected to the remote computer.

About this task

To extend the test coverage you can add devices from multiple mobile clouds by creating groups in the AFT Suite. The AFT Suite, then, can be used to run the tests simultaneously on devices connected to multiple mobile clouds.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Go to **File > New > Other**.

   The **Select a wizard** dialog is displayed.
3. Perform the following actions in the **Select a wizard** dialog:
   a. Select the **XML File** option in the **XML** folder.
   b. Click **Next**.

      The projects that contain the tests are displayed.
   c. Select the project where you want to save the XML file.
   d. Enter a name for the XML file in the **File name** field.
   e. Click **Next**.
   f. Select the **Create XML file from an XML template** option.
   g. Click **Finish**.

      An XML document is displayed for editing in the **Source** view tab.

      Note: Open the **Source** view tab if the **Design** view tab is displayed.

      A sample of the syntax or structure of the XML is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="<any_alphanumeric_value>" type="com.ibm.rational.test.ft.aftSuite">
 <group>
  <tests>
     <test path="<path_to_mobile_test_1>"/>
     <test path="<path_to_mobile_test_2>"/>
  </tests>
  <devices>
```

```
    <device id="<name of the device 1>”/>
  </devices>
  <locations>
    <location host="<location_where_test_is_run>"/>
  </locations>
 </group>
</inits>
```

Refer to the following table to know the variables for the attributes mentioned in the XML sample:

Element	Attribute	Variable	Description
inits	*id*	*<any_alphanumeric_value>*	Specifies that you can enter any alphanumeric value that helps to identify the XML file.
test	*path*	*<path_to_mobile_test_1><path_to_mobile_test_2><path_to_mobile_test_3>*	Specifies the path to the mobile test in your project. **Note:** You can right-click the test, and then go to **Properties > Resource** to find the path.
device	*id*	*<name_of_the_device_1>*	Specifies the name or id of the device that is connected to the device cloud.
location	*host*	*<location_where_test_is_run>*	Specifies the location as the computer that runs the test. The host can be the computer that is running Rational® Functional TesterHCL OneTest™ UI or the remote agent computer.

4. Create the following elements in the XML document:

Note: You can copy the sample syntax and then edit the values in the UI, or you can create the elements in the XML in the UI.

a. `<group>`

The start and end tags are automatically inserted.

You must create a separate group for each mobile clouds. Each group must contain the elements that are mentioned in in the following steps.

b. Create `<tests>` within the `<group>` tag.
c. Create `<test>` within the `<tests>` tag.
d. Add the attribute path to the `<test>` as follows:

```
<test path="">
```

Note: All values that you enter for the attributes must be enclosed within a double quotation mark.

e. Add additional rows for each mobile test.
f. Create `<devices>` within the `<group>` tag.
g. Create `<device>` within the `<devices>` tag.
h. Add the attribute id as follows:

```
<device id=""></device>
```

i. Create `<locations>` within the `<group>` tag.
j. Create `<location>` within the `<locations>` tag.
k. Add the attribute host as follows:

```
<location host=""></location>
```

5. Refer to the following table for the values of the attributes that you can use in the XML file:

Attribute	Example of a value		Explanation
test path	*MyProject\test_1*		The path in the workspace that contains the project with the test. You can find the path of the test from the **Test Navigator** by right-clicking the test, and then go to **Properties > Resource**.
Device id	BitBar	*Bitbar:Google Pixel 2*	The name of the Android device is Google Pixel 2 that is connected to the BitBar Cloud.
		Bitbar:Apple iPhone 6s Plus A1687 12.0.1	The name of the iOS device is Bitbar : Apple iPhone 6s Plus A1687 12.0.1 that is connected to the BitBar Cloud.
	pCloudy	*pcloudy:Pixel_2_API_30*	The name of the Android device is Pixel 2 API 30 that is connected to the pCloudy Cloud.
		pCloudy:Apple_iPhone8plus_-Ios_12.2.0_9db66	The name of the iOS device is Apple_iPhone8plus_Ios_12.2.0_9db66 that is connected to the pCloudy Cloud.
	Perfecto	*Perfecto:8D3E35CF16D8D827E4827ABBCD0E582E2761CADA*	The name of the Android device is 8D3E35CF16D8D827E4827ABBCD0E582E2761CADA that is connected to the Perfecto Cloud.

Attribute	Example of a value		Explanation
		Perfecto:R48904TNSAZ	The name of the iOS device is Perfecto : R48904TNSAZ that is connected to the Perfecto Cloud.
location host	*localhost*		The host is localhost if the device cloud is connected to the computer that runs Rational® Functional TesterHCL OneTest™ UI.
	1.20.30.40		The host is the IP address of the remote agent computer, if the device cloud is connected to the remote agent computer.

6. Enter the values for the attributes for your tests, device, and location in the XML file.
7. Save the XML file.

Results

You have created an AFT Suite for mobile tests, which you can use to run the mobile tests on devices that are connected to multiple mobile clouds.

What to do next

You can run the AFT suite on mobile devices connected to multiple mobile clouds. See Running mobile tests as an AFT Suite on multiple mobile clouds on page 1347.

Working with keywords

In Rational® Functional TesterHCL OneTest™ UI, you can record and then associate Test Workbench tests with the keywords that are created in IBM® Rational® Quality Manager. The keywords are used in manual tests in Rational® Quality Manager. When you run a manual test that contains a keyword associated with a Rational® Test WorkbenchHCL OneTest™ Studio test, the Test Workbench test is run on the workstation where the adapter has been started.

Viewing keywords

In Rational® Functional TesterHCL OneTest™ UI, you can record and then associate Test Workbench tests with the keywords that are created in IBM® Rational® Quality Manager. To view the keywords, you must log on to Rational® Quality Manager.

1. In the Keyword View, click **Logon to RQM**  .
2. In the **Logon to Rational Quality Manager** dialog box:

    a. Type the URL in the **RQM repository** field.
    For example: If Rational® Quality Manager is running in the local computer, type `https://localhost:9443/jazz` in the **RQM repository** field.

Note: If the Rational® Quality Manager server is renamed, make sure that you update the URL in the **RQM repository** field with the new server name.

b. Type a valid username and password in the **Username** and **Password** fields.

c. Click **Finish**.

3. Click **Refresh Keywords** . The connected projects keyword list is displayed in the keyword view.

a. To select a project area, click **ProjectArea** list.

Result

Only keywords pertaining to the selected project area are displayed.

b. Click **Get Keywords**.

a. **Optional:** To search for keywords that have a specific tag, type the tag in the **Search by tag** field.

b. **Optional:** Click **Get Keywords**.

Result

You can see the description of the keyword by expanding the keyword node. If the keyword is associated with a manual test, the steps from the manual test are displayed along with the keyword description.

Use the navigation buttons on the toolbar to get the next set of keywords and to move between pages.

Associating tests with keywords

You can automate an IBM® Rational® Quality Manager keyword by associating it with a Web UI test. You can either associate an existing test with a given keyword, or record a new test against the keyword and then associate the test.

Before you begin

The Keyword View must be displayed in the UI Test perspective to associate tests with keywords. To display the Keyword View, click **Window > Show View**. Select **Keyword View** under the **Test** category, and click **OK**.

Recording a test for a keyword

1. Click the keyword from the keyword view list to record a script and associate it with a Web UI test.
2. Right-click the keyword, and click **Record Test** .
3. Type a name for the test to be recorded in **Test name**.
   By default, the keyword name is used as the test script name.
4. Click **Finish**.
   Result
   The Recording monitor opens and the recording starts.

Associating an existing test with a keyword

About this task

You can only associate Web UI tests with a keyword. External tests like Selenium tests cannot be associated with a keyword. However, you can associate a compound test that contains a Selenium test with a keyword.

1. Click the keyword from the keyword view list to associate it with a Web UI test.
2. Right-click the keyword, and click **Associate Test**  .
   Result
   The **Select Test** dialog box lists the existing tests for the test projects.
3. Select the test that you want to associate from the list and click **OK**.
   Result
   The **Select Test Location and Channel** dialog box is displayed when you associate a test with the keyword for the first time. This dialog box specifies the location of the test project.
4. Do one of the following:
   - To indicate that the associated test must be run locally, select **Test will be run locally**.
   - To indicate that the test must be run from a shared location, select **Test will be run from the shared location**.
5. If you selected **Test will be run from the shared location**, type the path to the parent directory of the test project in the **Select the shared location** field, and ensure that you copy the project that contains the keyword-associated test to the specified shared location manually. For Rational® Quality Manager to run the test, the project with the test must be present in the shared location.
6. To select the channel of execution for the keyword-associated test, select the **Select Channel** check box and select the required channel from the list. If no channels have been defined in Rational® Quality Manager, the **Select Channel** check box and list are not available.
7. Click **Finish**.
   Result
   To view the tests associated with a keyword, right-click the keyword in the Keyword view and click **Show Associated Tests**.

Running manual tests that contain keywords from Rational® Quality Manager

When you run a manual test that contains a keyword from Rational® Quality Manager, the test associated with the keyword, if any, is run on the workstation on which the Rational® Quality Manager adapter is running.

About this task

You can either reference external resources by accessing a shared location or a local test machine. During execution of the tests, the test resources are copied to the test machine if it is a shared location. However, when you access resources in a local test machine, you are accessing resources on the path that you use.

1. In Rational® Quality Manager, create a test case.
2. Create a test script name and associate an existing test script with the test case in Rational® Quality Manager.
3. Execute the test case and view the results of the test execution after the playback. For information, refer the Rational® Quality Manager information center.

The Keyword view

The Keyword view displays the keywords created in IBM® Rational® Quality Manager. By default, this view is displayed along with the Protocol Data and Recording Control views in Rational® Test WorkbenchHCL OneTest™ Studio.

The following menu options are available when you right-click a keyword in the keyword view:

Refresh Steps

Lists the steps associated with a keyword.

Record Test

Records a Test Workbench test for the selected keyword.

Associate Test

Associates a Test Workbench test with the keyword. Multiple tests can be associated with a keyword.

Show Associated Tests

Shows all the tests associated with a keyword.

To view the keyword list:

- If the Keyword View is not displayed in the UI Test perspective, click **Window > Show View**. Select **Keyword View** from the **Type** category and click **OK**.

To select a project area:

- To select a project, click **ProjectArea** list. Only keywords pertaining to the selected project area are displayed.

Testing in the Functional Test perspective

When you develop Java, HTML, or Terminal-based applications, you can use IBM® Rational® Functional TesterHCL OneTest™ UI to create traditional functional tests for these applications. You must first record the tests and then use Rational® Functional TesterHCL OneTest™ UI to run the tests before you can view the test results.

You can find the following information:

- Preparing the functional test environment on page 774
- Managing functional test projects on page 858
- Working with functional test scripts (Windows-only) on page 865
- Verification points on page 904

- Driving functional tests with external data on page 937
- Functional test assets on page 864
- Testing terminal-based applications on page 956
- Troubleshooting issues on page 1042

Preparing the functional test environment

This section describes the tasks you must perform to configure the functional test environment.

You can refer to the following topics to learn more about preparing the functional test environment:

- Automatically enabled environment for functional testing on page 774
- Before you record on page 778
- Managing functional test projects
- Enabling Java environments on page 778
- Enabling web browsers on page 780
- Preparing for functional testing in Google Chrome browsers on page 784
- Enabling the Eclipse non-p2 based applications for functional testing on page 792
- Enabling the Eclipse p2- based applications for functional testing on page 793
- Enabling stand-alone Standard Widget Toolkit applications on page 795
- Enabling SAP client and server on page 805
    - Enabling SAP GUI scripting for Windows on page 805
    - Enabling the SAP server on page 806
- Enabling response time breakdown on page 808
- Browser enablement diagnostic tool on page 799
- Enabling the Java plug-in of a browser on page 800
- Configuring applications for testing on page 796
- Configuring Java environments for testing on page 797
- Configuring browsers for testing on page 798
- Creating and using a test object map to test applications
- Adding references to external resources on page 802
- Proxy settings for freeform datawindow PowerBuilder controls on page 803
- Functional Tester options

Automatically enabled environment for functional testing

From Rational® Functional TesterHCL OneTest™ UI version 8.2.2 9.1 onwards, Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing.

Typically, you prepare the functional test environment by enabling components such as browsers, associated Java™ Runtime Environments (JREs), Java™ plug-ins, and Eclipse platforms. With automatic enablement of the test environment, you can directly record and play back functional test scripts without enabling components manually.

Note:

- Automatic enablement option is available only on Windows machine.
- When you turn on automatic enablement, ensure that in addition to the application under test only the processes required by Rational® Functional TesterHCL OneTest™ UI are running on the computer.

To turn off automatic enablement in Rational® Functional TesterHCL OneTest™ UI Eclipse IDE version 8.5.1 9.1 and later, click **Window** > **Preferences** > **Functional Test**, then clear the **Automatic enablement** check box.

To turn off automatic enablement in Rational® Functional TesterHCL OneTest™ UI Visual Studio IDE version 8.5.19.1 and later, click **Tools** > **Options** > **Functional Test** the clear the **Automatic Enablement** check box.

Before version 8.5.1, automatic enablement is modified with the `rational.test.ft.browser.infest_on_demand` in the `ivory.properties` file. By default, this property is set to true.

The automatically enabled environment overcomes significant limitations that are seen while testing JRE versions later than Sun JRE 1.6 Update 17.

Rational® Functional TesterHCL OneTest™ UI enables the components automatically only in Microsoft™ Windows™ environments, including Federal Desktop Core Configuration (FDCC) setups. The automatic enablement takes place under certain conditions and has limitations. Table 1 lists the components that are enabled automatically and the components that need to be enabled manually. Table 2 lists the applications for which the test environment is enabled automatically, and the applications for which the environment must be enabled manually.

Table 8. Scenarios for automatic enablement - components

Component types	Automatically enabled	Enable manually
Browsers	• Automatic enablement is supported on Microsoft Internet Explorer for 32-bit Rational® Functional TesterHCL OneTest™ UI installation. **Learn more about automatically enabled browsers:** • Any associated Java™ plug-ins for the supported browsers are also enabled automatically. • You can test HTML applications that contain Sun JRE applets on automatically enabled 32-bit browsers. For this, you must replace the contents in the java.policy file by the following code to make sure that the browser is enabled automatically for applet testing:	• Google Chrome browsers • Microsoft™ Edge • Mozilla Firefox

Table 8. Scenarios for automatic enablement - components (continued)

Component types	Automatically enabled	Enable manually

	
	`grant {`
	`permission java.security.AllPermission;`
	`};`
	You must also make sure that during playback, the first click must be on the applet window.
	• Testing HTML applications that contain applets that are loaded in automatically enabled 64-bit browsers is not supported.	

JREs	All Sun or IBM® JREs versions 1.5 or later that are supported by Rational® Functional TesterHCL OneTest™ UI.	All Sun or IBM® JRE versions earlier than 1.5 that are supported by Rational® Functional TesterHCL OneTest™ UI

Table 9. Scenarios for automatic enablement - application domains

Automatically enabled environment	Enable environment manually
• HTML applications • Dojo applications • Java™ applications that contain Swing controls • Java™ Abstract Window Toolkit (AWT) applications • Java™ applications that are built by using Standard Widget Toolkit (SWT).  **Learn more about automatic enablement for SWT and 64-bit AWT applications:** Automatic enablement for SWT applications and 64-bit AWT applications has certain limitations and requires specific conditions. The test environment is enabled automatically if both Rational® Functional TesterHCL OneTest™ UI and the test applications use a JRE from the same vendor. If the JREs are from different vendors, complete one of the following steps so that the environment is enabled automatically:	• Adobe™ Flex applications • SAP GUI client and server, for testing SAP

Table 9. Scenarios for automatic enablement - application domains (continued)

Automatically enabled environment	Enable environment manually
• By default, Rational® Functional TesterHCL OneTest™ UI uses the IBM® JRE. Verify whether you can set the test application to use the IBM® JRE. If setting the test application to use the IBM® JRE is not possible, set Rational® Functional TesterHCL OneTest™ UI to use the Sun JRE that the test application uses. • If the test application uses Sun JRE, complete one of the following steps: ◦ Copy the `tools.jar` file and the `attach.dll` file from the Sun `jdk<version_number>` directory to the `jre<version_number>/lib/ext` directory. ◦ Start the test application with this command: `java -javaagent:"<Rational® Functional TesterHCL OneTest™ UI installation directory>\javaagent\FtAgent.jar"`	applications

Limitations and workarounds in automatically enabled environments

Automatically enabled test environments have the following limitations:

- For 64-bit Rational® Functional TesterHCL OneTest™ UI installer, automatic enablement is not supported for Internet Explorer browser. Dynamic enablement is supported only for 32-bit Rational® Functional TesterHCL OneTest™ UI installer for Internet Explorer browser.
- You cannot open the Verification Point Comparator by clicking the **View Results** link in the functional test HTML log. Instead, open the corresponding project log file from the functional test project log, in the Functional Test Projects view.
- When you test 32-bit SWT or Eclipse applications in automatically enabled environments, the first click action is not recorded. Perform the first click twice to make sure that it is recorded.
- In an automatically enabled test environment, if you uninstall a JRE that is associated with a browser, restart the computer, and then disable the uninstalled JRE in any browser add-ons that point to the uninstalled JRE, if any.
- In some combinations of JREs and operating systems, when the environment is automatically enabled, the browser shuts down unexpectedly when text is entered in a text box in an applet that is embedded in an HTML page. To resolve this, do one of the following procedures:
  - For Internet Explorer browsers, update the policy file with the permissions in the security folder of the JRE that is associated with the browser.
  - For Mozilla Firefox browsers, manually enable the browsers, and ensure that the Next-Gen plug-in is disabled.
  - Use the Scripting option to access elements relative to the enabled domain `toplevelwindow` (either `HTMLTopLevelWindow` or `JavaTopLevelWindow`)

Before you record

Before you can start recording functional test scripts, perform the following setup and configuration tasks:

- Create a functional test project. For an overview, see the related topic on Functional Test projects.

  Note: If you are part of a team, and the team already has a project set up, consider connecting to the existing project, instead of creating a new one. For more information, see the related topic on Connecting to a Functional Test project.

- Enable your browsers, Java environments and Eclipse platforms using the enabler.

  Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

  If required, you can manually enable or disable a browser, JRE or Eclipse platform, in the **Enable Environments** dialog box (the enabler).

- Configure the application for testing. See Configuring Applications for Testing on page 796.

Enabling Java environments

You need to enable Java™ environments before you can use Rational® Functional TesterHCL OneTest™ UI to test Java™ applications. Rational® Functional TesterHCL OneTest™ UI is shipped with a JRE that is automatically enabled during your installation. The JRE is called "Default JRE." To enable other JREs, or if you install a new JRE, you must run the enabler again.

Before you begin

To enable Java™ environments, you must log on as an administrator. On Microsoft® Windows® 7 operating system, you must also run Rational® Functional TesterHCL OneTest™ UI as an administrator.

About this task

In Rational® Functional TesterHCL OneTest™ UI, the automatic enablement check box is selected, which is the default option. When this option is enabled, the testing environment such as web browsers, the associated Java™ Runtime Environments (JREs), Java plug-ins, and Eclipse platforms are enabled automatically. Therefore, you can directly record the functional test scripts without enabling the testing environment manually. The automatic enablement of the testing environment takes place under certain conditions and limitations. For more information about the conditions and limitations on the automatic enablement, see the related links section.

1. Click **Configure > Enable Environments for Testing** any time from Rational® Functional TesterHCL OneTest™ UI to invoke the **Enable Environments** dialog box (the enabler). Click the **Java Environments** tab.
2. Click **Search.** The Search for Java™ Environments dialog box opens.

a. Select one of the following search mechanisms.
   - **Quick Search** can only be used on Windows® systems. It searches the Windows® registry for the Java™ environments, and is quicker than searching your hard disk drive(s)
   - **Search All Drives** scans all of your hard disk drives or partitions to locate all the Java™ environments on your system

   Note: You should not use the **Search All Drives** option to find JREs on Linux® systems. Instead use the **Search in** option and browse for the JRE.

b. Select **Search in** to browse to a specific drive or root directory to search.

c. After choosing one of the search mechanisms, click the **Search** button.

3. When the search is complete, Rational® Functional TesterHCL OneTest™ UI lists the JREs in the **Java Environments** list on the left side of the Java™ Environments tab. The list includes the full path name of each environment. Decide which environments you want to enable.
4. Select the environments you want to enable by clicking on them in the list. You can select multiple JREs by using the Ctrl key while selecting. Click the **Select All** button if you want to enable all of the JREs.
5. Click **Enable**.
   The selected environment(s) will be enabled for Java™ testing. The enabled environments will be indicated in parentheses after each JRE name in the list.
6. Select a JRE to be the default, and click the **Set as Default** button.
7. Click **Close**.

Results

Notes:

- **Enabling JVM:**

  When you run Rational® Functional TesterHCL OneTest™ UI for the first time, it automatically enables the JVM of your browser's Java™ plug-in so that HTML recording works properly. If you install a different JVM, you must rerun the enabler to enable it.

  However, if you experience an error regarding the Java™ plug-in during HTML testing, or when trying to launch the Verification Point Comparator from the HTML log, you need to make sure your plug-in is configured properly. See Enabling the Java Plug-in of a Browser on page 800 for instructions.

- You do not have to use the **Search** button to add an environment. You can click the **Add** button instead in Step 2. This brings up the Add Java™ Environment dialog box, which you can use to locate the new Java™ environment. After you select it and click **Add**, the environment will then be added to the **Java Environments** list, and you follow steps 4 - 7 to enable or disable it. If you try to add a file that is not a Java™ environment, you will get an error and it won't be added to the list.

- If your JRE is not enabled, you will be able to tell because the Recording Monitor on page 1807 will be blank when you try to record against a Java™ application. For this reason, leave the Recording Monitor in view while recording. If you see this symptom, you need to run the enabler.
- To enable browsers for HTML testing, see Enabling Web Browsers on page 780.
- You can test that your JRE is enabled properly by clicking the **Test** button in the enabler. This opens the JRE Tester, which reports the JRE version, JRE vendor, and whether the JRE is enabled successfully. Click **OK** to close the JRE Tester.

Upgrade Note: If you are upgrading Rational® Functional TesterHCL OneTest™ UI from earlier version versions, you might have to re-enable the Java environment. For more information, see upgrading from earlier versions of Rational® Functional TesterHCL OneTest™ UI topic.

Enabling web browsers

To test the HTML applications in Rational® Functional TesterHCL OneTest™ UI, you must enable web browsers. You can then record and play back tests by using the browsers that you enabled.

Before you begin

- You must have installed either 32-bit Java or 64-bit Java on your computer based on the web browser that you want to enable. See Enabling Java environments on page 778.
- You must have installed the 32-bit or 64-bit Oracle JRE to enable the Internet Explorer browser.

Restriction:

- For Guest users in Internet Explorer, with the **Protected mode** ON, recording and playback of functional test scripts do not work as expected.
- Recording on HTML dialog boxes that are embedded in other domains like Java, .NET and Windows is not supported. To perform actions on such embedded dialog boxes, edit the script manually using the `getScreen().inputKeys()` or `getScreen.inputChars()` API where required.
- On 64-bit operating systems, recording and playback on 64-bit Internet Explorer 11.0 browsers that are embedded in 64-bit Java, .NET or Windows or other applications are not supported.

- You must use 32-bit Rational® Functional TesterHCL OneTest™ UI for testing 32-bit Mozilla Firefox, Google Chrome, Microsoft Edge Browsers. Similarly, you must use 64-bit Rational® Functional TesterHCL OneTest™ UI for testing 64-bit Mozilla Firefox, Google Chrome, Microsoft Edge Browsers.

About this task

In Rational® Functional TesterHCL OneTest™ UI, the automatic enablement check box is selected, which is the default option. When this option is enabled, the testing environment such as web browsers, the associated Java™ Runtime Environments (JREs), Java plug-ins, and Eclipse platforms are enabled automatically. Therefore, you can directly

record the functional test scripts without enabling the testing environment manually. The automatic enablement of the testing environment takes place under certain conditions and limitations. For more information about the conditions and limitations on the automatic enablement, see the related links section.

1. Click **Configure > Enable Environments for Testing** in Rational® Functional TesterHCL OneTest™ UI.

   The **Enable Environments** dialog box is displayed.

2. Click the **Web Browsers** tab and perform the following steps:

   a. Click **Search**.

      The Search for Web Browsers dialog box is displayed.

   b. Search for the browser by using any of the following search methods:
      - Select **Search All** to find and list all the web browsers that are installed on your computer.

        Note: You must not use the **Search All** option to find the web browsers on a Linux or UNIX computer. Instead, you must use the **Search In** option and browse for the web browsers.

      - Select **Search In** to browse for all the web browsers that are installed in a specific directory.

   c. Click **Search**.

3. Select the web browsers that you want to enable from the **Web Browsers** list.

   Note: You can select multiple browsers by using the **Ctrl** key while you are selecting. Click **Select All** if you want to enable all of the browsers.

4. Click **Enable**.
   The word “enabled" is displayed along with the name of the web browser that is enabled.
5. Select a browser and then click **Set as Default** to set it as the default web browser.
6. Click **Close**.
7. Click **Test** to run the Browser Enablement Diagnostic Tool and verify whether the web browser is enabled correctly.

Results

The selected web browsers are enabled for testing the HTML applications.

What to do next

You can play back the Web UI tests by using the browser that you enabled.

Related reference

Automatically enabled environment for functional testing on page 774

Related information

Enabling the Google Chrome browser on page 787

Enabling Microsoft Edge to test HTML applications

You must enable the Edge browser before you record functional tests for HTML applications.

About this task

To enable the Edge browser, you must install the extension from the Chrome Web Store. Alternatively, you can also use the extension that is bundled along with Rational® Functional TesterHCL OneTest™ UI.

Note: You must preferably install the extension from the Chrome Web Store. You must use the extension bundled with the product only when you are unable to install it from the Chrome Web Store.

1. Open Rational® Functional TesterHCL OneTest™ UI.
2. Click **Configure > Enable Environments for Testing...**.

   The **Enable Environments** dialog box is displayed.
3. Add the Edge browser to the list of web browsers in the **Web Browsers** tab, if it is not added.

   To add the Edge browser, you must perform the following tasks:

   a. Click **Add**.

      The **Add browser** dialog box is displayed.

   b. Browse and select the `.exe` file of the Edge browser.
   c. Click **Apply**, and then click **Finish**.

   Result

   The Edge browser is added to the list of browsers under the **Web Browsers** tab.
4. Enable the Edge browser by performing the following tasks:

   a. Select the Edge browser in the **Web Browsers** tab, and then click **Enable**.

      A confirmation dialog box is displayed.

   b. Click **OK**.

      The Edge browser opens and the **Chrome Web Store** page is displayed.

   c. Click **Add to Chrome**.

      A confirmation dialog box is displayed.

d. Click **Add Extension**.

After the extension is added, the Rational® Functional TesterHCL OneTest™ UI icon is displayed in the browser toolbar.

5. Verify whether the Edge browser is enabled in Rational® Functional TesterHCL OneTest™ UI by clicking **Test** in the **Enable Environments** dialog box.

The test result is displayed as *Passed* if the Edge browser is enabled successfully.

Results

You have enabled the Edge browser to record functional tests for HTML applications.

What to do next

You can record a test by using the Edge browser. See Recording scripts to test HTML applications on page 883 for details.

Setting a specific browser profile for the playback of functional HTML tests

You can set a specific browser profile in IBM® Rational® Functional TesterHCL OneTest™ UI that includes the extensions and settings for the browser to run the functional HTML tests by using the Web UI engine.

About this task

You can set a specific browser profile for the following browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Note: When you set a specific browser profile for the Firefox browser, Rational® Functional TesterHCL OneTest™ UI creates a copy of the profile that you set and the tests are played back in the profile copy.

1. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.
2. Click the **Browser** tab.
3. Select the checkbox of the browser for which you want to set the profile path.

The **Browse** button for the browser is enabled.

4. Click **Browse** to browse and select the user profile for the browser.
5. Click **Apply and Close**.

Results

You have set a specific browser profile for the selected browser.

Related information

Playback of test scripts for HTML applications by using the Web UI engine on page 1447

Use the Profile Manager to create and remove Firefox profiles

Create a new browser user profile

Multiwindow support for functional HTML tests

When you record functional test scripts for HTML applications, Rational® Functional TesterHCL OneTest™ UI provides multiwindow support to record actions that you perform on multiple windows of the same browser instance.

While recording the test script, if a new browser window is displayed after you perform an action in the parent window of the application, Rational® Functional TesterHCL OneTest™ UI records actions on the new browser window also. Multiwindow refers to one or more new windows that open from the existing window depending on the user's action. When a new tab is opened in the same browser, then this new tab cannot be referred to as a multiwindow.

Rational® Functional TesterHCL OneTest™ UI provides multiwindow recording and playback support for the Google Chrome, Mozilla Firefox, and Microsoft Edge browsers. After you install or reinstall Rational® Functional TesterHCL OneTest™ UI, the **Play back with Web UI Extension** and **Play back with Web UI Action** checkboxes in **Windows > Preferences > Functional Test > Playback** are selected as the default options. When you play back a functional test script for an HTML application, Rational® Functional TesterHCL OneTest™ UI automatically selects the Web UI engine. Also, Rational® Functional TesterHCL OneTest™ UI finds controls from all the active windows of the browser instance.

Related information

Enabling web browsers on page 780

Google Chrome browser support on page 785

Preparing for functional testing in the Google Chrome browser

You can use Rational® Functional TesterHCL OneTest™ UI to test HTML applications in the Google Chrome browser. To do this, you must enable the Google Chrome browser and add the Rational® Functional TesterHCL OneTest™ UI extension for Google Chrome™ to the browser.

You can refer to the following topics to learn more about running tests in Google Chrome:

- Google Chrome browser support on page 785
- Enabling the Google Chrome browser on page 787
- Changing the web server port for communication with Google Chrome
- Troubleshooting functional tests in Google Chrome browsers on page 790

Google Chrome browser support

With IBM® Rational® Functional TesterHCL OneTest™ UI, you can test HTML applications that are loaded in the Google Chrome browser in Microsoft Windows environments.

You can test HTML applications that contain the following controls:

- HTML controls
- Dojo controls in applications built using Dojo Toolkit. For more information, see Dojo support on page 1709.

You can also perform the following actions:

- Test HTML applications that are loaded in multiple browser windows or embedded frames and inline frames, with varying zoom levels
- Record a functional test script in a Microsoft® Internet Explorer or Mozilla Firefox browser and play it back in a Google Chrome browser, provided that the Document Object Model (DOM) is compatible with the Google Chrome browser.
- Record a functional test script in a Google Chrome browser and play it back in a Microsoft® Internet Explorer or Mozilla Firefox browser, provided that the Document Object Model (DOM) is compatible with the other browsers.
- Use the manual scripting find() method while testing applications in a Google Chrome browser. To use the manual scripting find() method, a browser instance is required. The browser instance is returned only after the document is completely loaded and not when the browser starts.

Support for functional testing in the Google Chrome browser is version independent, so you can test HTML applications in any version of a Google Chrome browser.

The Rational® Functional TesterHCL OneTest™ UI Google Chrome™ extension

To perform functional testing in the Google Chrome browser, an extension is required to be added to the browser that enables communication between Rational® Functional TesterHCL OneTest™ UI and the Google Chrome browser through a web server. This extension, known as IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™, is available with your Rational® Functional TesterHCL OneTest™ UI installation. Alternatively, it is also available on the Google Chrome web store. To test applications loaded in Google Chrome, you must enable the browser.

Prerequisites for functional testing in the Google Chrome browser

Before you use Rational® Functional TesterHCL OneTest™ UI to test applications on the Google Chrome browser, complete these procedures:

1. In the Google Chrome browser settings, ensure that extensions are allowed, and that both Java and JavaScript are also allowed. Do not change the default browser settings.
2. Enable the browser manually for functional testing. To do this, complete one of the following steps:

- Enable the browser manually from the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI. The browser opens and you are prompted to add the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension to the browser. For instructions to enable the Google Chrome browser manually, see the related task named Enabling the Google Chrome browser.
- Add the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension to the browser from the web store. Verify that the browser is enabled by using the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI. For instructions to add the extension from the web store, see the related task named Adding the extension IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome. Ensure that you have an Internet connection to access the extension from the Google Chrome web store.

3. The default web server port for communication between Google Chrome and Rational® Functional TesterHCL OneTest™ UI is set on the Webserver Configuration page in the **Preferences** dialog box, as well as in the options for the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension. If this default port number is in use, you must specify an available port. Ensure that you specify the same port in both the **Preferences** dialog box as well as in the options for the extension.
4. Ensure that you start Rational® Functional TesterHCL OneTest™ UI before you start the application-under-test (AUT) in the Google Chrome browser. After you start Rational® Functional TesterHCL OneTest™ UI, open the AUT by completing one of the following steps:
   - Configure the AUT in the Application Configuration Tool in Rational® Functional TesterHCL OneTest™ UI and start the application from there. Alternatively, you can start the AUT at the time of recording by using the Start Application icon on the recording toolbar.
   - To start the Google Chrome browser independent of Rational® Functional TesterHCL OneTest™ UI, append `-allow-outdated-plugins -allow-file-access-from-files -always-authorize-plugins` to the Google Chrome shortcut, and then start the browser.

Points to remember while testing in the Google Chrome browser

- Record an action on an application page only after the document has loaded completely.
- Ensure that you start the browser only after you start either Rational® Functional TesterHCL OneTest™ UI or the recording monitor.
- To record and play back on local files that can be opened in a Google Chrome browser, ensure that you select the **Allow access to file URLs** check box in the extensionIBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™.

Troubleshooting functional tests in the Google Chrome browser

For useful information that will help you troubleshoot problems you face while testing in the Google Chrome browser, see Troubleshooting functional tests in the Google Chrome browser on page 790.or

Current limitations in Google Chrome testing:

- Recording on the Back and Forward buttons in the browser are not supported. Use browser-level `back()` and `forward()` APIs to play back these actions.
- When you record a script, actions on controls in dialog boxes are recorded as `click(atPoint())` relative to the dialog box. You can also use keystrokes to record on dialog box controls.
- Playback of actions on combo box drop-down controls is not supported. To play back these actions, modify the script manually and specify the option that must be selected during playback.
- The browser-level `deletecookies()` API is not supported. Cookies must be deleted manually.
- Recording on tabs is not supported.
- Recording on applications built using Dojo toolkit might be slow compared to other browsers such as Microsoft® Internet Explorer or Mozilla Firefox.
- Recording and playback is not supported in the Google Chrome browser in cases where the browser is started with a blank home page, that is, without a home URL.

Enabling the Google Chrome browser

To use Rational® Functional TesterHCL OneTest™ UI to test HTML applications that run on Google Chrome browser, you must enable the Google Chrome browser manually.

About this task

You must enable the browser for recording the tests. Rational® Functional TesterHCL OneTest™ UI facilitates playing back the tests directly in Chrome without enabling the browser.

You can enable the browser by using any of the following methods:

- By enabling the browser from the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI. The browser opens and you are prompted to add the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™ extension to the browser as documented in the following procedure.
- By adding the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension to the browser from the Google Chrome web store. You must ensure that you have an Internet connection to access the extension from the Google Chrome web store. After adding the extension, verify that the browser is enabled by opening the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI. For instructions to do this, see Adding the Google Chrome extension from the Chrome Web store on page 788.
- By installing the extension that is packaged with the product if you do not have access to internet to access the Chrome Web Store. For more information, see Adding the Google Chrome extension bundled with the product on page 789.

1. In Rational® Functional TesterHCL OneTest™ UI, click **Configure > Enable Environments for Testing** to invoke the **Enable Environments** dialog box (the enabler).
2. Click the **Web Browsers** tab. Add the Google Chrome browser to the **Web Browsers** list by doing the following:
    a. Click **Search**. The **Search for Web Browsers** dialog box opens.
    b. Search for the Google Chrome browser using any of the following search methods:

- Select **Search All** to let the enabler locate all the browsers on your system. Rational® Functional TesterHCL OneTest™ UI scans all the hard disk drives or partitions, and lists the browsers in the **Web Browsers** list.
- Select **Search In** to browse to a specific disk drive or root directory to search.

c. Click the **Search** button.

3. On the **Web Browsers** tab, click **Chrome** in the **Web Browsers** list, and click **Enable**.

   The Google Chrome browser opens and you are prompted to add the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension to the browser.

4. Click **Continue**, and then click **OK**.

   On the **Web Browsers** tab In the **Enable Environments** dialog box, the enabled status is indicated in parentheses after the Google Chrome browser name in the **Web Browsers** list.

5. Test that the browser is enabled properly by clicking the **Test** button to run the Browser Enablement Diagnostic Tool.
6. Click **Finish** In the **Enable Environments** dialog box.

Results

After you enable the Google Chrome browser, you can start testing applications that run in Google Chrome browsers.

Related information

ShowMe

Adding the Google Chrome extension from the Chrome Web store

To perform functional testing in the Google Chrome browser, an extension is required to be added to the browser that enables communication between Rational® Functional TesterHCL OneTest™ UI and the Google Chrome browser through a web server. This extension, known as IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™, is available with your Rational® Functional TesterHCL OneTest™ UI installation. Alternatively, the extension is also available on the Google Chrome web store. To test applications loaded in Google Chrome, you must enable the browser, which adds the extension to the browser.

Before you begin

- If you already enabled the Google Chrome browser manually through the **Enable Environments for Testing** dialog box in Rational® Functional TesterHCL OneTest™ UI, and you want to install the latest updates to the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension from the Google Chrome web store, first disable the browser by clicking **Disable** in the **Enable Environments for Testing** dialog box, and then add the extension from the web store by completing the procedure below.

About this task

1. In the Google Chrome browser, click the **Customize and Control Google Chrome** icon, and then click **Settings**.
2. On the Settings page, in the right pane, click **Extensions**.
3. Click **Want to browse gallery instead**. The Google Chrome web store opens.
4. In the **Search** field, type IBM® Rational® Functional TesterHCL OneTest™ UI and press Enter. The IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension is displayed.
5. Click **Add to Chrome**.

Results

The IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome extension is added to the Google Chrome browser, and the browser is enabled for functional testing.

What to do next

Verify that the browser is enabled by opening the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI. After you verify that the browser is enabled, you can start testing applications in the Google Chrome browser.

Related information

ShowMe

Adding the Google Chrome extension bundled with the product

To perform functional testing in the Google Chrome browser, you must install an extension for the Chrome browser. You can install the extension that is bundled with the product if you do not have access to the internet connection.

1. Go to the folder where the product is installed and follow these steps:
    a. Navigate to the following path: `FunctionalTester/bin/enabler`.
    b. Extract the `RFTChromeExtension.crx` file to the local directory.
2. Perform following steps in the Google Chrome browser:
    a. At the top right corner, click **More > Extensions**.
    b. Turn on the **Developer Mode**.
    c. Click the **Load Unpacked** button to browse and select the folder that you extracted.
3. Click **Add to Chrome**.

Results

The Google Chrome extension is added to the Google Chrome browser, and the browser is enabled for functional testing.

Changing the web server port for communication with Google Chrome

The default web server port for communication between Google Chrome and Rational® Functional TesterHCL OneTest™ UI is set on the Webserver Configuration page in the **Preferences** dialog box in Rational® Functional TesterHCL OneTest™ UI, as well as in the options for the Rational® Functional TesterHCL OneTest™ UI for Google

Chrome™ extension. If this default port is already in use on the workstation where you are testing applications in Google Chrome, change it and specify an available port.

Before you begin

- Ensure that you have enabled the Google Chrome browser for functional testing.
- The web server port is used when you enable the Google Chrome browser manually in the **Enable Environments for Testing** dialog box. By default, the port 9100 is set for the web server. If this port is already in use, change it and specify an available port.



Note: If you change the port in the Options for the extension, ensure that you also make the same change on the Webserver Configuration page on page 856 in the Rational® Functional TesterHCL OneTest™ UI **Preferences** dialog box. For more information, see the related topic named Webserver Configuration page.

1. In the Google Chrome browser, click the **Customize and Configure Chrome** icon.
2. In the left pane, click **Extensions**.
3. Under the extension IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™, click the **Options** link.
4. In the **Webserver Port** field, change the default port (9100) and specify a different port that is available on the workstation. If you change the port in the Options for the extension, ensure that you also make the same change on the Webserver Configuration page on page 856 in the Rational® Functional TesterHCL OneTest™ UI **Preferences** dialog box.
5. Select the **Allow access to file URLs** check box if you intend to test local files that can be opened in a Google Chrome browser.
6. Click **Save**.

Related information

ShowMe

Troubleshooting functional tests in the Google Chrome browser

If you encounter problems while testing in the Google Chrome browser, you will find useful information in this section to resolve them.

- The Google Chrome browser is not properly enabled on page 791
- It is not possible to record on a Google Chrome browser on page 791
- Actions are recorded on the Windows domain and not in the application domain on page 791
- Problems with playing back certain actions on page 791

The Google Chrome browser is not properly enabled

If the error CRFCN0794E on page 1667 is displayed when you try to record on a Google Chrome browser, the browser is not properly enabled. This can be due to one of the following reasons:

- No Sun Java Runtime Environment (JRE) was associated with the browser, or the associated JRE was not enabled. To resolve this, associate Sun JRE 1.6 Update 10 or later with the Google Chrome browser, and then verify that the browser has been enabled by opening the **Enable Environments** dialog box in Rational® Functional TesterHCL OneTest™ UI.
- The default web server port (9100) for communication between the Google Chrome browser and Rational® Functional TesterHCL OneTest™ UI is being used by another application on the workstation. Change the default port and specify an available port in both the Webserver Configuration page on page 856 in the Rational® Functional TesterHCL OneTest™ UI **Preferences** dialog box, and in the Options for the IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™ extension. For instructions to do this, see the related topics Changing the web server port for communication with Google Chrome and Webserver Configuration page.



Note: Ensure that you specify the same port number in both places.

It is not possible to record on a Google Chrome browser

This problem could occur due to one of the following reasons:

- The browser was not properly enabled. Ensure that the browser is properly enabled.
- The browser was started with a blank home page, that is, without a home URL. To prevent this, always specify a home URL for the Google Chrome browser.

Actions are recorded on the Windows domain and not in the application domain

While recording on the Google Chrome browser, actions on objects in the test application may be recorded in the Windows domain and not in the application domain. This can occur due to one of the following reasons:

- The action was recorded before the document was loaded completely in the Google Chrome browser. To prevent this, record an action on an application page only after the document has loaded completely.
- The browser was started before either Rational® Functional TesterHCL OneTest™ UI or the recording monitor was started. To prevent this, always start the browser only after you have started either Rational® Functional TesterHCL OneTest™ UI or the recording monitor.
- Actions on controls in dialog boxes are recorded as `click(atPoint())` relative to the dialog box. To prevent this, use keystrokes to record on dialog box controls.

Problems with playing back certain actions

Some actions recorded on the Google Chrome browser need the script to modified, to be played back successfully. For example

- Clicking the Back and Forward buttons in the browser. To play back these actions, modify them in the recorded script using the browser-level `back()` and `forward()` APIs.
- Actions on combo box drop-down controls. To play back these actions, modify the script manually and specify the option that must be selected during playback.

:

Enabling the Eclipse non-p2 based applications for functional testing

If the application under test is a non-p2 Eclipse based application, you can enable the Eclipse platform for functional testing using the Eclipse enabler.

Before you begin

Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

1. Click **Configure > Enable Environments for Testing** fromRational® Functional TesterHCL OneTest™ UI.
2. Click the **Eclipse Platforms** tab.
3. Click **Search**. You can search for the Eclipse platforms in two ways:
   Choose from:
   - **Search all drives**: Searches the Eclipse platforms in all your hard disk drives
   - **Search in**: Searches the Eclipse platform in the specific directory
4. Select a search mechanism and click **Search**.
   Result
   The search results are listed in the left pane of Enable Environments window.

   Note: Use the **Add** button to browse to the Eclipse platform and add directly.

5. Select the Eclipse platform that you want to enable from the left pane of the window. The detailed information consisting of the name and path of the selected Eclipse platform is displayed in the right pane of the window.
6. To enable support for GEF, select the **GEF support** checkbox.
   Result
   The GEF enablement plugin is copied to the plugin directory of the AUT.
7. Click **Enable**.
   Result
   The selected Eclipse platform is appended with *Enabled* in parentheses.
8. Click **Finish**.
9. If the application under test (AUT) is an Eclipse based rich client platform (RCP) application that shell shares with Rational® Functional TesterHCL OneTest™ UI, do this:

a. Click **Configure > Configure Applications for Testing** fromRational® Functional TesterHCL OneTest™ UI.

b. Click **Add** to add the Eclipse application, and click **Next**.

c. Select **Executable file** and click **Next**.

d. Browse to the application executable file, and click **Open**.

e. Click **Finish**.

The application is listed in the **Applications** list in the Application Configuration Tool.

f. In the **Args** field, specify `-vmargs -Dft.testability=true`.

g. Click **Run** to start the application.

Note: You must start an AUT that is an Eclipse based RCP application that shell shares with Rational® Functional TesterHCL OneTest™ UI with the arguments `-vmargs -Dft.testability=true`, from the Application Configuration Tool.

What to do next

Configuring applications for testing on page 796

Related information

Enabling the Eclipse p2- based applications for functional testing on page 793

Enabling the Eclipse p2- based applications for functional testing

If the application-under-test is based on p2-based Eclipse, you must enable the Eclipse platform for testing using the Eclipse Software Updates feature.

Before you begin

Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

Enabling applications based on Eclipse version 3.4 and later

You can enable applications that are based on Eclipse version 3.4 and later using the Eclipse Software Updates feature.

1. Open the application under test.
2. Click **Help > Software Updates**.
3. Click the **Available Software** tab.
4. Click **Add Site**.
5. Click **Local**.
6. Browse to the **EclipseEnabler** directory under the `FunctionalTest` folder in the product installation location.
7. Click **OK**.

Enabling applications based on versions prior to Eclipse version 3.4

You can enable applications that are based on Eclipse versions prior version 3.4 for testing using the Eclipse Software Updates feature.

1. Open the application under test.
2. Click **Help > Software Updates > Find and Install**.
3. Select **Search for new features to install** in the Feature Updates dialog box and click **Next**.
4. Click **New Local Site** in the Update sites to visit page.
5. Select the **EclipseEnabler** directory found under the `FunctionalTest` folder in the product installation location. Click **OK**.
6. Specify the name for the local site in the **Name** field and click **OK**. The site name is listed in the Update sites to visit page.
7. Select the site from the list and click **Finish**. The specified directory, along with its sub folders, is listed in the Search Results page.
8. Select the main directory in the Search Results page and click **Next**.
9. In the Feature License page, select **I accept the terms in the license agreement** and click **Next**.
10. Click **Finish**.

    You can verify that the Eclipse application is enabled for testing by checking that the com.rational.test.ft.enabler.wsw plugin is added to the Plugins directory of the Eclipse application.

    Note: Some Eclipse based RCP applications may not have the **Help > Software Updates** option. In this case you can copy the com.rational.test.ft.enabler.wsw_7.0 plugin manually from the FunctionalTest \EclipseEnabler\Plugins directory from the product installation location into the Plugins directory of the RCP application. After doing this, restart the RCP application with the -clean option.

What to do next

Configuring applications for testing on page 796

Related information

Enabling the Eclipse non-p2 based applications for functional testing on page 792

Enabling stand-alone Standard Widget Toolkit applications

You must enable stand-alone Standard Widget Toolkit (SWT) support before using Rational® Functional TesterHCL OneTest™ UI to test SWT applications in the application under test.

About this task

To enable an SWT application, you must first enable the JRE in which the application runs, and then modify the Java code of the SWT application.

Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

1. Enable the JRE in which the SWT application runs. To do this:
    a. Click **Configure > Configure Applications for Testing** from Rational® Functional TesterHCL OneTest™ UI to invoke the Enable Environments dialog box.
    b. Click the **Java Environments** tab.
    c. Click **Search**. The Search for Java Environments dialog box opens.
    d. Select the appropriate search mechanism, and click **Search**.
    Result
    When the search is complete, the JREs are listed in the **Java Environments** list.
    e. Select the environment of the SWT application by clicking it on the list.
    f. Click **Enable** and then click **Close**.
2. Place the rational_ft_bootstrap.jar file in the classpath. The rational_ft_bootstrap.jar is found in the `C:\Program Files\IBM\SDP\FunctionalTester\EclipseEnabler\plugins` `C:\Program Files\HCL\HCLOneTest\FunctionalTester\EclipseEnabler\plugins` location.
3. The enableSwtUi() method must be called from the User Interface (UI) thread of the SWT application. Add this code:

```
try
 {
 com.rational.test.ft.bootstrap.Bootstrap.enableSwtUi(this);
 }
 catch (Throwable e) {}
```

    This must be called from the code that first creates the application shell.
4. Save your changes.

Configuring applications for testing

You must configure your Java™, HTML, VB.NET, SAP GUI, Flex or Windows® applications for functional testing by providing the name, path, and other information that Rational® Functional TesterHCL OneTest™ UI uses to start and run the application. You use the Application Configuration Tool to configure applications.

Before you begin

Important: If you enabled Mozilla Firefox or Google Chrome browser for IBM® Rational® Functional TesterHCL OneTest™ UI, the latest Java update must be associated with the browser. If not done, security messages prompt up when you open the browser and Java will be blocked.

To add and configure your applications:

1. Open Rational® Functional TesterHCL OneTest™ UI and click **Configure > Configure Applications for Testing**. You can also open the Application Configuration tool from the Start Application dialog box by clicking the **Edit** button.
2. To add a new application, click the **Add** button.
3. In the Add Application dialog box, select the application type, and click **Next**.
4. Select the type of application and the path of the application to enable it for functional testing.
   For different types of domains, specify the application details as specified in the following table:

   Table 10.

Domain type	Application details
Java	Click **Browse > Open** to find and select the application. The file types can be with .class or .jar extensions.
Visual Basic .Net and Windows®	Click **Browse > Open** to select any executable or batch file.
HTML	Select either **Local** or **URL**. If **Local**, browse to an .htm or .html file. If **URL**, enter the URL address.
SAP	Select the SAP executable from the dropdown list. Alternatively, you can select a SAP shortcut file with a .exe, .sal or .sap extension using the **Browse** button. **Note:** You must have SAPGUI installed in your computer to be able to select this option.

   Note: To configure Flex applications for testing, see Configuring Flex application using the user interface on page 814.

5. Click **Finish**.

The application will then show up in the **Applications** list in the Application Configuration Tool.

6. Look at the information in the **Detailed Information** list.
   - For Java™ applications, the **Name**, **Kind**, **Path**, **.class/.jar file**, and **Working Dir** fields are automatically filled in for you. The **JRE**, **Classpath**, and **Args** fields are optional, and could be filled in by you as needed. Make any necessary edits.
   - For HTML applications, the **Name**, **Kind**, and **URL** fields will be filled in. Select the default browser in the **Browser** field.
   - For SAP applications, you can specify a separate argument to use in the **Args** field.
7. To test if the application is configured properly for testing, select the added application from the list and click **Run**.
8. Click **OK** to save the changes you made.

Note: You can edit the information about an application in the tool at any time.

Configuring the Java environment for testing

You must configure JREs for testing Java only when the application under test is either Internet Explorer or a Java-based application that requires a Java distinct from the Java which is bundled with the product. The configured Java provides path, run options, and other information that Rational® Functional TesterHCL OneTest™ UI needs to access and use your JREs.

1. Click **Configure> Enable Environments for Testing** in Rational® Functional TesterHCL OneTest™ UI.
2. Click the **Java Environments** tab.

   You can use this tab to add and edit JRE configurations.

3. Add your JRE(s) by using one of the following methods:
   - Perform the following steps to add a new JRE by using the **Search** method:
     a. Click the **Search...** button.

        The **Search for Java Environments** dialog box is displayed.

     b. Choose one of the search options, and then click the **Search** button.

        Note: You must not use the **Search All Drives** option to find JREs on Linux® or UNIX® systems. Instead, you must use the **Search In** option and browse for the JRE. See Enabling Java Environments on page 778 for information on the search options. The details of the JRE are populated automatically, except for the **Run Options** field.

   - Perform the following steps to add a new JRE by using the **Add** method:
     a. Click the **Add** button to add a new environment manually.

        The **Add Java Environment** dialog box is displayed.

     b. Browse and select the JRE that you want to add.

You can select any directory under the root of the JRE, or the root directory itself.

c. Click the **Add** button.

The JRE is displayed in the **Java Environments** list.

4. Look at the detailed information of the selected JRE that is displayed at the right-hand side.

Note: Whether you use **Search** or **Add**, the detailed information on each JRE is filled in, except for the **Run Options** field. You can edit the details.

For information on fields of the **Java Environments** dialog box, see the related links section.

5. Select the JRE, which you want to set as the default environment for the playback, from the list, and then click the **Set as Default** button.

The default JRE that you set is indicated in parentheses. You can change the default JRE any time by using this tab. You can also override the default environment for a specific application, by indicating it in the **JRE** field in the **Application Configuration Tool** dialog box. For more information, see the related links section.

6. Click **OK** or **Apply** to save the changes that you made.

Note: After a JRE is added, you can edit the information any time by opening this tab and selecting it in the **Java Environments** list.

What to do next

You can start recording tests for the application under test.

Related reference

Application Configuration Tool on page 1735

Configuring browsers for testing

You need to configure your browsers for HTML testing with Rational® Functional TesterHCL OneTest™ UI. To configue browsers, you must provide name, path, and other information about the browsers.

1. Click **Configure > Enable Environments for Testing** from Rational® Functional TesterHCL OneTest™ UI.
2. Click the **Web Browsers** tab.
   This tab is used to add and edit browser configurations.
3. Add your browser(s) by using the Search on page 798 or Add on page 799 button:

Search

a. Click the **Search** button to add a new browser by searching.

This opens the **Search for Web Browsers** dialog.

b. Choose one of the search options in that dialog, and then click the **Search** button.

Note: You should not use the **Search All** option to find browsers on Linux® or UNIX® systems. Instead, use the **Search In** option to locate the browser. Rational® Functional TesterHCL OneTest™ UI displays all the information on each browser.

Add

a. Click the **Add** button to add a new browser manually.

The **Add Browser** dialog appears.

b. Locate the directory containing the browser you want to add.
c. With the file selected, click the **Add** button.

The browser appears in the **Web Browsers** list.

4. Read the information in the **Detailed Information** list.
Whether you use **Search** or **Add**, Rational® Functional TesterHCL OneTest™ UI displays all the information on each browser. Make any necessary edits.
5. Select a browser from the list, and then click the **Set as Default** button.

The selected browser is set as the default browser, and it is indicated in parentheses. The default browser is used for testing all HTML applications that have not specified a browser. You can change the default browser any time by using the **Web Browsers** tab.

6. Click **Finish** or **Apply** to save the changes.

Note: Once a browser has been added, you can edit its information any time by selecting it in the **Web Browsers** list.

Related reference

Web Browsers tab on page 1871

Related information

Enabling Web Browsers on page 780

Browser enablement diagnostic tool

The Browser Enablement Diagnostic Tool is used to diagnose problems you might have with enabling your browser for HTML testing. The tool will diagnose the enablement problem and report how to solve the problem.

About this task

Use the diagnostic tool if you suspect that HTML is not being tested properly. If you are trying to record against an HTML application, and nothing shows up in the Recording Monitor, the browser is probably not enabled properly. It

might mean that the Java™ plug-in of your browser is not enabled. If that is the case, the diagnostic tool will tell you how to enable the browser. The tool offers quick and simple directions to solve any problem it finds.

To run the tool:

1. Open the Rational® Functional TesterHCL OneTest™ UI Enabler by clicking **Configure > Enable Environments for Testing**.
2. Click the **Web Browsers** tab.
3. Click the **Test** button. The Browser Enablement Diagnostic Tool opens.
4. Click the **Run Diagnostic Tests** button.

Results

About this task

The **Results** page tells you whether the test passed or failed. If the test failed, this page will also list the problem.

Problem and solution

About this task

The **Problem and Solution** page will list the problem and explain how to solve it. Follow the instructions listed there and close the tool. If you were in the process of recording a script when you ran the tool, stop recording the script and start over. The recording should then work against an HTML application.

Details (Advanced)

About this task

The **Details** page list additional information about your environments. The **Java Enabled** field indicates whether Java™ is enabled in your browser. The **JVM Information** field lists information about your JVM. The **General Enablement Information** field lists Java™ and HTML domain information.

Enabling the Java plug-in of a browser

The Oracle Java™ plug-in of your browser(s) must be enabled in order for some applets to be tested, and for the **View Results** link that launches the Verification Point Comparator from the HTML log to work properly. If you get an error regarding the plug-in during HTML testing, or when trying to launch the Comparator, use the following steps to fix the problem.

About this task

Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.

Note:

- The JRE that is shipped with Rational® Functional TesterHCL OneTest™ UI and is set as the default, is not configured with the browser. If you do not have another JRE on your system that includes a browser plug-in, you must install one before using Rational® Functional TesterHCL OneTest™ UI. Download a version of J2SE that contains a JRE and a browser plug-in. You need a simple Java™ runtime and not a JDK or Java™ desktop. After installing the JRE, follow the steps below to enable the plug-in.
- Java plugin can be enabled in Java 8 and earlier but it is not supported in Java 9 and later.

1. Configure your plugin to make sure your Oracle Java™ plug-in is configured properly on your system.
    a. Open your Windows® Control Panel.
    b. Double-click the **Java Plug-in** icon to open the Java™ Plug-in Control Panel.

       If you have more than one plug-in listed, use the one that is version 1.4 or later.
    c. In the Java™ Plug-in Control Panel, click the **Browser** tab.
    d. Select Firefox and Microsoft® Internet Explorer.
    e. Click **Apply**.
    f. Close the Java™ Plug-in Control Panel.
    g. Close the Windows® Control Panel.
2. Re-enable all your Java™ environments in Rational® Functional TesterHCL OneTest™ UI.
    a. Disable any currently enabled environments. In Rational® Functional TesterHCL OneTest™ UI, click **Configure > Enable Environments for Testing** to open the Enable Environments dialog box (the enabler). Click the Java Environments tab on page 1785.
    b. Click the **Select All** button beneath the Java™ Environments list to select all current Java™ environments.
    c. Click **Disable** to disable them.
    d. Click **Search.** The Search for Java™ Environments dialog box opens. Select one of the following search mechanisms.

       Quick Search can only be used on Windows® systems. It searches the Windows® registry for the Java™ environments, and is quicker than searching your hard disk drive(s).

Search All Drives scans all of your hard disk drives or partitions to locate all the Java™ environments on your system.

Note: You should not use the **Search All Drives** option to find JREs on Linux® or UNIX® systems. Instead use the **Search in** option and browse for the JRE.

Select **Search in** to browse to a specific disk drive or root directory to search.

e. After choosing one of the search mechanisms, click the **Search** button.

f. Select the environments you want to enable by clicking on them in the **Java Environments** list. Click the **Select All** button beneath the list to enable all of them.

g. Click **Enable**.

h. If you want to change the default JRE, select your preferred default, and click the **Set as Default** button.

Note: You can test that your browser plug-in is enabled properly by clicking the **Test** button in the Web Browsers tab of the enabler. This opens the Browser Enablement Diagnostic Tool on page 799. If you suspect your browser is not enabled properly, run the diagnostic tool and follow the instructions it gives to solve the problem.

Adding references to external resources

Functional test scripts or projects may refer to or use external resources like the DLLs in .Net IDE or JAR files in Eclipse IDE. To enable the scripts or projects to use these files, you must add references to these files in Rational® Functional TesterHCL OneTest™ UI.

Adding references to functional test Java project

1. Copy the JAR files in the Functional Test customization folder. By default, the folder is available at `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\OneTestUI\customization` in Windows and at `/etc/opt/IBM/Rational/RFT/customization` `/etc/opt/HCL/OneTestUI/customization` in Linux.
2. Verify whether the copied JAR files are referred by the functional test project.
   - Open Rational® Functional TesterHCL OneTest™ UI, right-click the functional test project and click **Properties**.
   - In the Properties page, click **Java Build Path**. In the Libraries page, verify whether the added JAR file is listed in the `RFT Customization Libraries. OneTest UI Customization Libraries.`
   - If the `RFT Customization LibrariesOneTest UI Customization Libraries` is not displayed, right-click the functional test project and click **Reset Java Build Path**.

Adding references to functional test VB.Net project

Copy the DLL files in the Functional Test customization folder. By default, the folder is available at `C:\Documents and Settings\All Users\Application Data\IBM\RFT\customization` `C:\Documents and Settings\All Users\Application Data\HCL\RFT\customization` in Windows.

For referring the assembly files that are registered in Global Assembly Cache (GAC), you must add these DLL file names in the **NetAssembliesForCompiler.txt** file available at `Functional Test install directory\FunctionalTester\bin` location.

Proxy settings for freeform DataWindow PowerBuilder controls

In Rational® Functional TesterHCL OneTest™ UI, from version 8.2.1 onwards, new proxies are available for recording and playback on freeform DataWindow PowerBuilder controls in functional test scripts. By default, in a Rational® Functional TesterHCL OneTest™ UI 8.2.1 installation, scripts that you recorded using the old proxies can be played back normally but any new scripts that you record will use these new proxies.

Depending on your requirement, you can revert to using the pre-8.2.1 proxies for all scripts or use the new proxies for the scripts. The usage of proxies for recording and playback on freeform DataWindow Powerbuilder controls in functional test scripts are controlled by these flags in the `ivory.properties` file:

- `rational.test.ft.pb.datawindow.freeform.usenewproxyonly`
- `rational.test.ft.pb.datawindow.freeform.usehybridproxy`

Table 11. Flag settings for controlling proxies

Flag in the `ivory.properties` file	Default setting in Rational® Functional TesterHCL OneTest™ UI 8.2.1	Setting to use only the pre-8.2.1 proxies	Setting to use only the new proxies
`rational.test.ft.pb.datawindow.freeform.usenewproxyonly`	FALSE	FALSE	TRUE
`rational.test.ft.pb.datawindow.freeform.usehybridproxy`	TRUE	FALSE	Setting ignored

The default setting for the two flags in a Rational® Functional TesterHCL OneTest™ UI 8.2.1 installation is shown in the first column of the table. On playing back a script, the `GetChildren` function looks for the accessibility object references and the new proxies added in version 8.2.1. By default, the new proxies are used only when you record a new script.

To revert to using only the pre-8.2.1 proxies, set the `rational.test.ft.pb.datawindow.freeform.usehybridproxy` flag in the `ivory.properties` file to FALSE. Recording and playback are reverted to using the pre-8.2.1 proxies. This setting is shown in the second column of the table.

To use only the new proxies that were added in version 8.2.1, set the `rational.test.ft.pb.datawindow.freeform.usenewproxyonly` flag in the `ivory.properties` file to TRUE. This setting is shown in the third column of the table. It ignores the `rational.test.ft.pb.datawindow.freeform.usehybridproxy` flag setting, and uses only the new proxies for recording and playback. This setting is useful if you do not need to play back any existing scripts and if you are recording only new scripts. It is also the setting which ensures the most optimal playback performance.

Note: The new proxy settings apply only to standard Win32 targets that are deployed by using PowerBuilder.

Setting up the environment for testing AJAX-based web applications

You can test AJAX-based applications in two different ways; by setting the Auto Trace option to true or by setting the Auto Trace option to false. If you set the Auto Trace option to false, you must use the Rational® Functional TesterHCL OneTest™ UI APIs for AJAX in the script by manually inserting them.

Setting the Auto Trace option to true

About this task

To test AJAX-based applications with the Auto Trace option set to true:

1. Open the ivory.properties file available at: <**Rational® Functional TesterHCL OneTest™ UI installation directory>\Functional Tester\bin**.
2. Set the `rational.test.ft.html.ajax.autotrace` option to true.
   For example: `rational.test.ft.html.ajax.autotrace = true`

Setting the Auto Trace option to false

About this task

To test AJAX-based applications with the Auto Trace option set to false:

1. Open the ivory.properties file available at: <**Rational® Functional TesterHCL OneTest™ UI installation directory>\Functional Tester\bin**.
2. Set the `rational.test.ft.html.ajax.autotrace` option to false.
   For example: `rational.test.ft.html.ajax.autotrace = false`
3. In the recorded script, insert the `setAjaxTrace(true)` method for the required Document control to trace the AJAX requests. For example, `document_htmlDocument().setAjaxTrace(true)`
4. Use the `GetAjaxPendingRequests ()`, `WaitForAjaxPendingRequests (int)`, `GetAjaxCompletedRequests ()`, or `WaitForAjaxCompletedRequests ()` methods explicitly in the script to trace the AJAX requests.

What to do next

Note: During playback, if any action invokes an AJAX request, the subsequent action is performed only after the request is completed. If the AJAX request completion time is more than the script playback timeout value, use the `waitforExistence()` method for the Document control.

Enabling AJAX support for a pre-existing script

The HTML Document control is mapped to GuiTestObject in Rational® Functional TesterHCL OneTest™ UI versions prior to 7.0.0.2. The GuiTestObject does not contain any AJAX-related APIs. To use the AJAX-related APIs, the HTML Document control must be mapped to DocumentTestObject.

1. Open the functional test script that needs to be modified for enabling the AJAX support.
2. Click **Script > Open Test Object Map**. Select the Document control from the list and click **Administrative** tab.
3. Replace the **Test Object Class Name** value from **GuiTestObject** to **DocumentTestObject**. Save and close the Test Object Map.
4. Click **Script > Update Script Helper**

   Result

   Note: By default, the HTML Document control is mapped to DocumentTestObject for enabling AJAX support. You can disable AJAX support and map the HTML Document control to GuiTestObject by setting the `rational.test.ft.html.use.documenttestobject = false` in the ivory.properties file.

Enabling SAP client and server

To use Rational® Functional TesterHCL OneTest™ UI to test SAP applications, you must enable the SAP client and SAP server.

Enabling SAP GUI scripting for Windows

To use Rational® Functional TesterHCL OneTest™ UI to test SAP GUI for Windows applications, enable the SAP GUI scripting using the SAP GUI enabler. The **SAPgui** tab option is not available on Linux.

Before you begin

Ensure that you have administrator privileges to use the SAP GUI enabler.

1. Open Rational® Functional TesterHCL OneTest™ UI and click **Configure > Enable Environments for Testing**.
2. Click the **SAPgui** tab.

   Note:
   - If you do not have the supported version of SAP GUI client in the Windows operating system, the **SAPgui** tab is disabled.
   - The **SAPgui** tab is not available on Linux.

3. Click **Enable**. The enabled state is displayed in the State field.

4. Click **Test** to verify that the SAP GUI client scripting is enabled. A message that the scripting is successfully enabled for SAP GUI is displayed.
5. Click **OK** and then click **Finish** .



Enabling SAP scripting in SAP GUI: Alternatively, you can enable the SAP scripting in the SAP GUI application.

a. Start the SAP Logon and log in to the SAP server.
b. Click **Customize Local Layout > Options**
c. In the **Options** window, select the **Scripting** tab.
d. Select the **Enable scripting** check box.
e. Clear the **Notify When a Script Attaches to a Running GUI** check box and the **Notify When a Script Opens a Connection** check box.
f. Save the settings and restart the SAP GUI.

What to do next

You must enable the SAP server for testing.

Enabling the SAP server

After you enable the SAP GUI client, enable the SAP server for testing by setting up scripting temporarily from the SAP client.

About this task

The value that you set with this procedure is lost when the server is restarted.



Note: If the server administrator edits the application server profile of the SAP system to include `sapgui/user_scripting = TRUE`, scripting is enabled by default when the server is restarted.

To enable the SAP server for enabling the scripting:

1. Start the SAP Logon and log in to the SAP server.
2. Start a RZ11 transaction.
3. Type `sapgui/user_scripting` in the Maintain Profile Parameters window.
4. Click **Display**.
5. Click **Change value** in the Display Profile Parameter Attributes window.
6. Type `TRUE` in the **New value** field.
7. Save the settings and log off from the SAP GUI.
8. Exit the SAP Logon program.



Tip: In SAP you can change the network connection mode to any server. The two connection modes are: High Speed Connection (LAN) and Low Speed Connection (Reduced Network Traffic). Although, functional testing works in both the modes, a script recorded using High Speed Connection plays back

only in that mode. This is also true for the Low Speed Connection mode. You must play back your SAP script in the same network connection mode at which you recorded. High Speed Connection mode provides the best results, because it provides the most valid recognition properties.

Note: The SAP server scripting setting is temporary. If the SAP server is restarted, you must again enable the SAP server for testing.

What to do next

You can now create a functional test project and start recording scripts to test the application.

Enabling applications with WebDynPro controls for functional testing

When you playback functional test scripts to test HTML applications with WebDynPro controls, you may get object not found exception or a weak recognition warning even if the control exists.

Perform the following tasks to make it easier for testing HTML applications with WebDynPro controls:

1. Close Rational® Functional TesterHCL OneTest™ UI and open the ivory.properties file located at `<product installation directory>\SDP\FunctionalTester\bin<product installation directory>\HCLOneTest\FunctionalTester\bin`.
2. Set `rational.test.ft.html.enabledynamicallyignoreidorname=true`.
3. Save and close the file.

Enabling SAP support for pre-existing Rational® Functional TesterHCL OneTest™ UI projects

To use the SAP support for a project that was created using the releases of Rational® Functional TesterHCL OneTest™ UI version 6, you must add two new templates (one for script headers and one for script helper headers) to the project. You will also need to update the build or reference information for the project.

1. In the Functional Test Projects view, right-click the project and select the **Properties** option.
2. In the Properties dialog box, select **Functional Test Script Templates** from the navigation list.
3. Select the template type **Script Helper: Header of the file**.
4. If you have not customized this template, you can upgrade it by clicking the **Restore Defaults** button.
5. Add the line `import com.rational.test.ft.object.interfaces.SAP.*;` in the import section of the template.
6. After modifying the template, click the **Apply** button.
7. Select the template type **Script: Header of the file** and add the same line in the import section of the template.
8. Finally, right-click the project again in the Functional Test Projects view and click **Reset Java Build Path**.

   Each member of your team must perform this last step, as the Java Build Path is local to each project on each machine.

Enabling the GEF application

You must enable the GEF support before using Rational® Functional TesterHCL OneTest™ UI to test GEF objects on the application under test.

About this task

To enable the GEF application:

1. Click **Configure > Enable Environments for Testing**.
2. Click the **Eclipse Platforms** tab.
3. Search for the Eclipse platform.

   Result

   The search results are listed in the left pane under Eclipse platforms.
4. Select the Eclipse platform that you want to enable.
5. Select the **GEF Support** check box.

   Note: If you have enabled an Eclipse platform without GEF support and want to enable support for GEF:
   a. Select the Eclipse platform and click **Disable**.
   b. Select the **GEF support** check box.
   c. Click **Enable**.

   Result

   The GEF enablement plugin is copied to the plugin directory of the AUT.
6. Click **Finish**.

Enabling response time breakdown

You can enable response time breakdown to see how much time is spent in each part of the application as the test runs. To collect response time breakdown, the data collection infrastructure must be installed and running on all computers that are used in the distributed application under test.

About this task

To enable response time breakdown:

1. From the product menu, click **Window > Preferences**.
2. Expand **Functional Test > Playback**.
3. Click **Response Time Breakdown** in the left pane.
4. Select **Enable Response Time Breakdown** check box.
5. Click **OK**.

Enabling Response time breakdown during playback

You can enable the response time breakdown during playing back the script.

About this task

To enable response time breakdown during playback:

1. Run the script.
   Result
   The Select Log window is displayed.
2. Type the log name for your script and click **Next**.
   Result
   The Specify Playback Options window is displayed.
3. Select **Enable Response Time Breakdown** check box.
4. Click **Finish**.

Flex applications testing process

The testing process is based on the tasks that Flex developers and testers perform. The process for automating functional tests of Flex applications differs, depending on the way developers create the application under test.

Automated testing of Flex applications requires you to load supporting files. These supporting files can be loaded in two different stages:

- At compile time for applications that are enabled for functional testing
- At run time for applications that are not enabled for functional testing

Prerequisites

Review the detailed list of information about IBM® Rational® Functional TesterHCL OneTest™ UI system requirements.

To test Flex applications, be sure the following software is installed:

Development environment

- Adobe Flex SDK and Adobe Flex automation framework
- Adobe Flex Builder

Test environment

- One of the following browsers:
  - Google Chrome
  - Microsoft Internet Explorer
  - Mozilla Firefox
- Adobe Flash Player ActiveX control

Note: A detailed list of the supported product versions is available here: Software Product Compatibility Reports

Assumptions

The following assumptions apply for the testing of Flex applications:

- Testers are not skilled in developing Flex applications.
- Testers cannot access Flex source code, the Flex compiler, or Flex documentation.
- Flex developers do not know how to use Rational® Functional TesterHCL OneTest™ UI.

Testing Flex applications

The tasks that you perform in testing Flex applications depend on the application and whether you are a developer or a tester.

You can test applications that are enabled for Rational® Functional TesterHCL OneTest™ UI or test applications that are not enabled for Rational® Functional TesterHCL OneTest™ UI.

Testing Rational® Functional TesterHCL OneTest™ UI enabled Flex applications

Developers can enable the Flex applications for testing by compiling the Flex application with Rational® Functional TesterHCL OneTest™ UI agent (rftFlex4.5.swc and rftProp_Flex4.5.swc) and Flex automation framework libraries. After compilation, the developer must create an HTML wrapper that embeds the enabled Flex application and provide the application on a web server or on a local test computer for testing.

Advantages and limitations of testing Rational® Functional TesterHCL OneTest™ UI enabled Flex applications

Three advantages encourage enabling Flex applications for functional testing:

- Efficiency: Multiple enabled Flex applications can be embedded in a single HTML page and can be tested simultaneously.
- Ease: Testing is simplified when different Flex applications communicate with each other. All the related enabled Flex applications can be embedded in a single page and can be tested based on a single scenario.
- Location: Enabled Flex applications can be tested locally.

One limitation in testing enabled Flex applications is that only the developer can enable the Flex application for testing.

Testing nonenabled Flex applications

Developers can enable the runtime loader component for Flex applications and deploy the application on a web server for testing.

Advantages and limitations of testing non-enabled Flex applications

The advantages of testing Flex applications that are not enabled for functional testing:

- Ease: Testers benefit because many technical complexities are hidden.
- Efficiency: Load and test multiple SWF files.

Review the following limitations regarding nonenabled Flex applications:

- Deployment option: The Flex application can be deployed in a test or production environment.
- Location: The runtime loader cannot be run locally; you must deploy the runtime loader to a web server.

Setting up the development environment for Flex applications

The developer must set up the development environment before enabling the Flex application for testing with Rational® Functional TesterHCL OneTest™ UI.

Consider the following points when you set up the development environment for Flex:

- The automation framework is a part of Flex Builder for Flex 4.5.
- For data visualization in Flex 4.5, make sure that you also include the `datavisualization.swc` file for these versions that is available on the Adobe site.
- For Spark controls in Flex 4.5, make sure that you also include the `automation_spark.swc` file for these versions that is available on the Adobe site.

Setting up the test environment for testing Flex applications

A correct environment setup for testing Flex applications helps ensure reliable functional testing results. The test environment is typically set up by the testers on the test computer.

Before you begin

Note: Ensure that the msvcp71.dll is available in the System32 directory (`C:\Windows\System32`) or SysWow64 directory (`C:\Windows\SysWOW64`).

About this task

To set up the test environment:

1. Check the settings of the browser:
    a. Internet Explorer:
        i. Open Internet Explorer.
        ii. Click **Tools > Internet Options**.
        iii. Click the **Security** tab.
        iv. Select the appropriate web content zone. Do one of the following steps:

- If the web server is configured on a remote machine, complete these steps:
    1. Select **Local Intranet**.
    2. Click **Sites > Advanced** .
    3. In the **Add this Web site to the zone** field, type the URL to add to the web server.
    4. Click **Add**, and then click **OK**.
- If the web server is configured on local host, complete these steps:
    1. Select **Local Intranet**.
    2. Click **Custom Level**.
    3. In the **Reset to** list, select **Medium-low**.
    4. In the Settings pane, click **Enable** for **Initialize and script ActiveX controls not marked as safe**.
    5. Click **OK**.

b. Firefox:

i. Click **Tools > Options > Content**.

ii. Clear the **Block pop-up windows** check box.

iii. Click **Tools > Options > Security**.

iv. Clear the **Warn me when sites try to install add-ons** check box.

2. Open Rational® Functional TesterHCL OneTest™ UI, and click **Configure > Enable Environments for Testing**. You must enable the required JRE and set Internet Explorer as the default web browser.
3. Enable the required browser on the Web Browsers page.
4. Make the application under test trusted to run the application locally. Paths to individual files or directories can be trusted, rendering all the files in each selected directory and any of its subdirectories trusted. Follow these steps to assign trust designations:
    a. Create a folder, `FlashPlayerTrust`, in `C:\WINDOWS\system32\Macromed\Flash`.
    b. Create a file named `Flex` without any file extension in the `FlashPlayerTrust` folder.
    c. Type the directory path of the Flex application in the `Flex` file. For example, if the Flex application is in `C:\Test` directory, type the path in the `Flex` file as `C:\Test`.
    d. Save the file.



Note: If you are testing a Flex 4.5 application, ensure to specify security settings in Flash Player.

Security Settings for Adobe Flex 4.5

About this task

Specify security settings only if you are testing a Flex 4.5 application.

1. Open an application in Flash Player.
2. Right-click and select **Settings** to access **Settings Manager**.
3. Select the **Privacy** tab.
4. Click **Advanced**.

Result

Adobe Flash Player launches a new browser window and loads the Settings Manager help page.

5. Click **Global Security Settings** panel link.
   Result
   The **Global Security Settings** window opens.
6. Add your application directory into secured or trusted directory. In the **Always trust files in these locations** drop down menu, click **Add location**. Browse for the location.

Note: For more information about setting the security configuration, see the Adobe® website.

Testing Rational® Functional TesterHCL OneTest™ UI enabled Flex applications

Developers can enable the Flex applications for testing by building the Flex application with Rational® Functional TesterHCL OneTest™ UI agent and Flex automation framework libraries. After compilation, the developer must create an HTML wrapper that embeds the enabled Flex application and provide the application on a web server or on a local test computer for testing.

To test Flex applications :

1. A developer must perform these tasks:
   a. Set up the development environment.
   b. Compile the Flex application with the Rational® Functional TesterHCL OneTest™ UI agent and Flex automation framework libraries. You can use either the Flex user interface or any of the following tools to enable the Flex application:
      - Flex Builder
      - Command-line compiler
   c. Create an HTML wrapper that embeds the enabled Flex application (.swf file) using an <object> and <embed> tags.
   d. Deploy the enabled Flex application for testing to a web server or provide the files for testing locally.
2. A tester must perform the following tasks:
   a. Set up the test environment.
   b. Obtain the enabled .swf file and HTML wrapper from the developer.
   c. Deploy the application to a web server or run the Flex application on a local test computer.
   d. Start Rational® Functional TesterHCL OneTest™ UI to test the HTML application that contains the embedded Flex application.

Configuring Flex applications

You can configure the Flex application with the Rational® Functional TesterHCL OneTest™ UI agent and Flex automation framework libraries.

About this task

You can use either the Rational® Functional TesterHCL OneTest™ UI Flex user interface or one of the following tools to enable the Flex application:

- Flex Builder
- Command-line compiler

Configuring Flex application using the user interface

You can configure the Flex application for functional testing using the user interface.

Configuring a Web application at compile time

You can configure the Web application during compile time using the UI.

About this task

To configure the web application during compile-time:

1. Click **Configure > Configure Applications for Testing**.
2. Click **Add** in the Application Configuration Tool window.
3. Select **Flex Application**, and click **Next**.
4. Select **Configure Flex application setup**, and click **Next**.
5. Select the **Web application** as the type of Flex application.
6. Select **Compile-time** from the **Enablement type** list.
7. Select the Flex version from **Flex SDKs** list.
8. Click **Browse** to select the application that has a .as or .mxml extension in the **Application** field.
9. Select the **Dependency Files** and **Additional Libraries** check box, if required, and click **Add** to select dependency and library files.
10. Click **Browse** to select a location for the **SWF Target Location** field.
11. Select the **Generate HTML Page** check box, if required.
12. Type the application URL in the **Specify Web Application URL** field, and click **Finish**.

Configuring a local application at compile time

You can set up your Flex application locally only during compile time.

About this task

To configure the local application during compile time:

1. In the Flex Application configuration window, select **Local application** as the type of Flex. The Enablement type is set to **Compile-time** by default.
2. Select the Flex version from the Flex SDKs list.
3. Click **Browse** to select the application that has an .as or .mxml extension in the **Application** field.
4. Select the **Dependency Files** and **Additional Libraries** check box, if required, and click **Add** to select dependency and library files.
5. Click **Browse** to select a location for the **SWF Target Location** field.
6. Select the **Generate HTML Page** check box, if required.
7. Click **Finish**.

Configuring Flex application using tools

You can configure your Flex application for functional testing using the tools like command-line compiler and Flex Builder.

Using the command-line compiler to enable Flex applications

Developers can compile a Flex application with the IBM® Rational® Functional TesterHCL OneTest™ UI agent and Flex Automation Libraries from a command line and enable the application for functional testing.

About this task

Compile the Flex application using the Rational® Functional TesterHCL OneTest™ UI agent and Flex Automation Libraries by running the following command.

To compile and enable a Flex 4.5 application for functional testing:

At a command line, type the following command, and press Enter:

```
"flash builder installation directory\Flash Builder 4\sdks\4.5.0\bin\mxmlc.exe"-include-libraries+="flash builder installation directory\Flash Builder 4\sdks\4.5.0\frameworks\libs\automation.swc;flash builder installation directory\Flash Builder 4\sdks\4.5.0\frameworks\libs\automation_agent.swc;flash builder installation directory\Flash Builder 4\sdks\4.5.0\frameworks\libs\datavisualization.swc;flash builder installation directory\Flash Builder 4\sdks\4.0.0\frameworks\libs\automation_spark.swc;flash builder installation directory\rftFlex4.0.swc;functional tester installation directory\rftProp_Flex4.0.swc" Test.mxml
```

Note:

- The above command has Flex 4.5 SDK deployed at *flash builder installation directory*\Flash Builder 4\sdks\ with directory as 4.5.0.
- The `automation_spark.swc` file has been included for Spark controls.

Note: The `datavisualization.swc` component is separately available at the Adobe site.

The default Flex Builder installation directory on Windows is `C:\Program Files\Adobe`.

This command is also available as a batch file with Rational® Functional TesterHCL OneTest™ UI installed. Testers can provide this batch file to the developer to enable the Flex application for testing. Provide the flex application source code filename as the parameter to the batch file.

The batch file is available in *product installation directory*\Functional Tester\Flex folder.

Exemple

For example, if your .mxml file is *Test.mxml*, the command to run the batch file is as follows:

For Flex 4.5:

```
buildapplicationwithFlex45adaptor.bat Test.mxml
```

or

```
buildapplicationwithFlex45adaptor64bit.bat Test.mxml
```

Using Flex Builder to enable Flex applications

Developers can use the Flex Builder to make Flex applications ready for functional testing.

About this task

Flex Builder is useful for developers who want to develop Flex applications (.swf files) that are ready for functional testing. To set up Flex Builder to support Flex automation and functional testing:

1. Start Flex Builder.
2. Create a new Flex project.
3. Select the Flex project in the navigator.
4. Click **Select Project > Properties > Flex Compiler**.
5. Type the following argument in the **Additional compiler arguments** field:

   For Flex 4.5:

   Note: You can include the `automation_spark.swc` for Spark controls.

   ```
   -include-libraries flash builder installation directory\Flash Builder
    4\sdks\4.5.0\frameworks\libs\automation.swc? flash builder installation directory\Flash Builder
    4\sdks\4.5.0\frameworks\libs\automation_agent.swc? flash builder installation directory\Flash
    Builder 4\sdks\4.5.0\frameworks\libs\datavisualization.swc? flash builder installation directory\Flash
    Builder 4\sdks\4.5.0\frameworks\libs\automation_spark.swc functional tester installation
   directory\rftFlex4.5.swc functional tester installation directory\rftProp_Flex4.5.swc
   ```

   Elements in italics are variables and depend on your directory structure.
6. Click **OK** to save your changes.
7. Click **OK**.

   Result

   The Properties dialog box closes.
8. Compile your Flex application.

Creating an HTML wrapper

The developer creates an HTML wrapper after compiling the Flex application with the Rational® Functional TesterHCL OneTest™ UI agent and Flex automation testing libraries.

About this task

The wrapper embeds the .swf file in an HTML page by using the <object> and <embed> tags.

You can use the default HTML wrapper file that the compiler generates along with the Flex application or use the HTML wrapper file in the functional testing application folder.

After creating the HTML wrapper, the developer passes the application and the HTML wrapper to the testers for functional testing. Typically developers deploy the files to a web server that testers access.

The following example shows how the Flex application Test.swf is embedded in the HTML page:

Exemple

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
id="myapp" width="100%" height="100%"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab">
  <param name="movie" value="Test.swf" />
  <param name="quality" value="high" />
  <param name="bgcolor" value=#CCCCCC />
  <param name="allowScriptAccess" value="sameDomain" />
  <embed src="Test.swf" quality="high" bgcolor=#CCCCCC
   width="1000" height="500" name="myapp" align="middle"
   play="true"
   loop="false"
   quality="high"
   allowScriptAccess="sameDomain"
   type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer">
   </embed>
</object>
```

Providing enabled Flex applications for testing

Developers can deploy a Flex application that is enabled for Rational® Functional TesterHCL OneTest™ UI for testing on a web server or provide the files for testing locally.

About this task

After enabling the Flex application for testing, perform either of these steps:

- Provide the .swf file and HTML wrapper to the testers so that they can test the Flex application on a local test computer.
- Deploy the Flex application to a web server that testers can access, and request that the testers test the Flex application from the provided URL.

Testing an enabled Flex application that is deployed on a web server

Testers can test a Flex application that is enabled for Rational® Functional TesterHCL OneTest™ UI and is deployed on a web server.

About this task

The developer deploys the Flex application to be tested on a web server and provides the URL to the testers. The testers can also deploy the files, provided that the testers can access the web server and the application files.

To test a web-based Flex application that is enabled for Rational® Functional TesterHCL OneTest™ UI:

1. Ensure that the application is added in the Application Configuration tool, enable the required JRE, and set your default web browser.
2. Obtain the enabled .swf file and HTML wrapper from the web server.
3. Open the HTML page in a browser.
4. Start Rational® Functional TesterHCL OneTest™ UI
5. Start testing the HTML file that contains the embedded Flex application.

Testing enabled Flex applications on a local test computer

Testers can test a Flex application that is embedded in an HTML wrapper on a local test computer.

Before you begin

Before testing a Flex application locally, verify that the .swf file is trusted. Also ensure that you add the application in the Application Configuration tool, enable the required JRE and set your default web browser.

About this task

To test locally the Flex application that is enabled for Rational® Functional TesterHCL OneTest™ UI:

1. Request the HTML wrapper from the file system.
2. Open the HTML page in the browser.
3. Start Rational® Functional TesterHCL OneTest™ UI.
4. Start testing the HTML application that contains the embedded Flex application.

Test Flex applications that are not enabled using Rational® Functional TesterHCL OneTest™ UI

Developers can enable the RuntimeLoading component for Flex applications and provide the application on a web server for testing.

To test Flex applications :

1. A developer must perform the following tasks:
    a. Set up the development environment.
    b. Configure the RuntimeLoading application.
    c. Deploy the RuntimeLoader files to a web server. Typically deploy the files to the root directory so that this action is performed only once.
    d. Deploy the Flex application to a web server.
    e. Provide the complete URL of the RuntimeLoading testing page to the testers.
2. A tester must perform the following tasks:
    a. Set up the test environment.
    b. Open the RuntimeLoadingTest.html page in browser.
    c. Type the relative path of the Flex application to be tested as a query parameter to the HTML page.
    d. Use Rational® Functional TesterHCL OneTest™ UI to test the application.

Note: Testers can also deploy the necessary files such as RuntimeLoader and the .swf file to a web server, provided that testers can access the server. Although developers might not be familiar with Rational® Functional TesterHCL OneTest™ UI, the developer needs the RuntimeLoader source file and the Rational® Functional TesterHCL OneTest™ UI agent (rft.swc) to enable and deploy the Flex application. Testers need to provide these files to the developer.

Configuring the non-enabled Flex application

You can configure the Flex applications that are not compiled using theRational® Functional TesterHCL OneTest™ UI agent and Flex automation libraries.

About this task

The developer must first set up the development environment before enabling the Flex application for testing with Rational® Functional TesterHCL OneTest™ UI by configuring the Runtime loader to generate the SWF file for the application to be tested.

After the SWF file is generated, the Flex application must be configured to create the URL that Rational® Functional TesterHCL OneTest™ UI must use to test the SWF file.

1. To generate the SWF file for the Flex application, complete these steps:

   Note: Runtime loaders for the supported Flex versions such as 4.5 can be configured by providing the appropriate SDK libraries and changing the output filenames accordingly. The runtime loader files that are enabled corresponding to the SDK under use, must be embedded in the html page.

   a. Compile the *RuntimeLoader.mxml* to swf file using the command: `"C:\Program Files\flex\FlexSdk4.5\4.5.0\bin\mxmlc.exe" -include- libraries+="C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation.swc;C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation_agent.swc;C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation_dmv.swc;%IBM_RATIONAL_RFT_INSTALL_DIR%\rftFlex4.5.swc;%IBM_RATIONAL_RFT_INSTALL_DIR%\rftProp_Flex4.5.swc" - output="C:\Program Files\IBM\SDP\FunctionalTester\Flex\RuntimeLoaderFlex30.swf" "C:\Program Files\IBM\SDP\FunctionalTester\Flex\RuntimeLoader.mxml".` `"C:\Program Files\flex\FlexSdk4.5\4.5.0\bin\mxmlc.exe" -include- libraries+="C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation.swc;C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation_agent.swc;C:\Program Files\flex\FlexSdk4.5\4.5.0\frameworks\libs\automation\automation_dmv.swc;%HOTUI_INSTALL_DIR%\rftFlex4.5.swc;%HOOTUI_INSTALL_DIR%\rftProp_Flex4.5.swc" - output="C:\Program Files\HCL\HCLOneTest\FunctionalTester\Flex\RuntimeLoaderFlex30.swf" "C:\Program Files\HCL\HCLOneTest\FunctionalTester\Flex\RuntimeLoader.mxml".`

Note: For Flex 4.5, you can also include the `automation_spark.swc` file for Spark controls.

b. Open the *RuntimeLoadingTestFlex30.html* with notepad.

c. Replace the movie source in the object and embed tags with *RuntimeLoaderFlex30.swf*.

2. To create the URL that is to be used to test the SWF file, complete these steps:
   a. Click **Configure > Configure Applications for Testing**.
   b. Click **Add** in the Application Configuration Tool window.
   c. Select **Flex Application**, and click **Next**.
   d. Select **Configure Flex application setup** in Flex application configuration type, and Click **Next**.
   e. Select **Web Application** as the type of Flex application.
   f. Select **Runtime** from the **Enablement type** list.
   g. Select the Flex version from the **Flex SDKs** list.
   h. Click **Browse** to select the SWF file of the application to be tested in the **Application** field.
   i. Click **Finish**.

Deploying the Runtime loader components

You must deploy the Runtime loader components to test Flex applications that are not compiled using the Rational® Functional TesterHCL OneTest™ UI agent and Flex automation framework libraries.

About this task

You can use the following file for configuring and testing Flex 4.5 applications:

- RunTimeLoaderFlex45.swf and RuntimeLoadingTestFlex45.html

Note: Testers can also deploy the necessary files to a web server, provided that testers can access the server. Although developers might not be familiar with Rational® Functional TesterHCL OneTest™ UI, the developer needs the Runtime loader source file and the Rational® Functional TesterHCL OneTest™ UI agent to enable and deploy the Flex application. Testers need to provide these files to the developer.

To deploy the Runtime loader components and enable testing of applications that are not compiled using Rational® Functional TesterHCL OneTest™ UI agent and Flex automation framework libraries, perform the following steps:

1. Deploy the *RuntimeLoadingTest.html* and *RuntimeLoader.swf* files from the `functional tester installation directory\FunctionalTester\Flex` directory to the web server.
2. Provide the URL of RuntimeLoadingTest.html page to the testers, for example, `http://localhost/RuntimeLoadingTest.html?automationswfurl=applicationtotest.swf`, where *applicationtotest.swf* is the name of the Flex application to be tested.

Testing nonenabled Flex applications

Testers can test a Flex application that is not enabled for functional testing and is deployed on a web server.

About this task

Developers deploy the application on a web server and provides the URL to the testers. Testers must perform the test using the Runtime Loader component that is available with Rational® Functional TesterHCL OneTest™ UI.

Important: If you encounter problems while the Flex application is loading using the Runtime Loader test page, check the security settings of Internet Explorer ActiveX and plug-ins.

Ensure that you add the application in the Application Configuration tool, enable the required JRE and set your default web browser.

To test a nonenabled Flex application:

1. Copy the compiled RunTimeLoader.swf and the RuntimeLoadingTest.html to the folder that contains the Flex application to the server.
2. Open the RuntimeLoadingTest.html page in browser.
3. Pass the relative path of the Flex application to be tested as a query parameter to the HTML page. For example, type:`http://localhost/RuntimeLoadingTest.html?automationswfurl=applicationtotest.swf`, where *applicationtotest.swf* is the file name of the Flex application.
4. Start Rational® Functional TesterHCL OneTest™ UI.
5. Start testing the HTML file that contains the embedded Flex application.

Importing and exporting configuration and customization files

You can configure and customize files in the configuration and the customization directories. The configuration files contains information such as the application that must be configured for testing. However, the customization files contain information about the external jar files used in your project, customization of any recognition properties, and third-party proxy extensions. You can export these files and later deploy them using the export and import utility. The default location for the configuration and customization file is `C:\ProgramData\IBM\RFT` `C:\ProgramData\HCL\OneTestUI`. For example: If you want to use the same configuration and customization files on different computers, you can archive these files and later deploy them using this utility.

Exporting the configuration and customization files

You can export the current configuration and customization file into a jar file. This jar file can be imported or deployed to any computer.

To export the resources to an archive file:

1. Click **File > Export**.
   Result
   The Export wizard opens.

2. Click **Functional Test > Functional Test configuration/customization to a JAR file**. Click **Next**.
3. Select the configuration and customization items to export. In the **File** text field, type the file name. You can also click **Browse** to select the destination path. Click **Finish**.
   Result
   The configuration and customization file is exported to the specified location.

Importing the configuration and customization files

The import wizard imports the configuration (.rftcfgjar), and customization (.rftcust) files to the Rational® Functional TesterHCL OneTest™ UI configuration, and customization directory. The import wizard displays the available items and the user can import these items into Rational® Functional TesterHCL OneTest™ UI or any computer. The configurations file (.rftcfgjar) is either merged with an existing configuration file or is replaced. The customization files are only replaced and not merged. You must restart Rational® Functional TesterHCL OneTest™ UI to activate some customization files.

About this task
To import items from an archive file:

1. Click **File > Import**.
   Result
   The Import wizard opens.
2. Click **Functional Test > Functional Test configuration and customization items**.
3. Click **Next**.
   Result
   The **Import configuration items** window opens.
4. In the **Import from** field, browse for the archive file in the file system. You must browse for the .rftcfgjar file.
   Result
   The **Select the items to be imported** window opens.
5. The items that the archive file contains are displayed in the **Select the items to be imported** window. Select the items to import and click **Finish**.
   Result
   The selected items are now displayed in Rational® Functional TesterHCL OneTest™ UI.

Setting preferences

You use the Preferences dialog box to customize Rational® Functional TesterHCL OneTest™ UI in a number of different areas, such as settings for time options; colors for the Verification Point Editor, the Verification Point Comparator, and the Object Map Editor; highlight color for test objects; operating system; playback; delays; log; playback monitor; ScriptAssure(TM); recorder; recording monitor; and the workbench.

To change preferences for the current user:

1. From the Rational® Functional TesterHCL OneTest™ UI menu, click **Window > Preferences**.
2. In the Preferences dialog box, in the left pane, expand **Functional Test**.
3. Select the appropriate preferences page for the options you want to change.

4. Change the options.

   In some cases, you might want to clear the **Use Default** field to edit the option.

5. Click **Apply** to save the new setting and continue changing options or click **OK** to save the new setting and close the Preferences dialog box.

Rational® Functional TesterHCL OneTest™ UI Preferences

You use the Preferences dialog box to customize various aspects of Rational® Functional TesterHCL OneTest™ UI, such as settings for time options; colors for the Verification Point Editor, the Verification Point Comparator, and the Object Map Editor; highlight color for test objects; operating system; playback; delays; log; playback monitor; ScriptAssure(TM); recorder; recording monitor; and the workbench.

To display the Rational® Functional TesterHCL OneTest™ UI Preferences page, click **Window > Preferences** and expand **Functional Test** in the left pane.

Use the Rational® Functional TesterHCL OneTest™ UI preferences pages to set options in the following areas:

- Rational® Functional TesterHCL OneTest™ UI on page 828
- Colors on page 830
- Highlight on page 831
- Operating System on page 836
- Playback on page 836
    - Delays on page 839
    - Logging on page 834
    - Monitor on page 840
    - ScriptAssure(TM)-Standard on page 841
    - ScriptAssure(TM)-Advanced on page 840
    - Enabling the unexpected window handling feature on page 842
- Recorder on page 843
    - Monitor on page 844
- Workbench on page 857
    - Advanced on page 858

Editing the preferences pages changes the current user profile only. It does not change settings for all users.

You can change the color settings on page 826 of information in the Verification Point Editor, Verification Point Comparator, and Object Map. Rational® Functional TesterHCL OneTest™ UI changes the color of elements in the editor. You can also change the fonts on page 828 for information in the .NET IDE. The colors and fonts feature is useful for enhancing the accessibility for people who have physical challenges, such as restricted mobility or limited vision. For more information, see the Colors Page on page 830.

Note: Some Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration preferences and Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration options are shared by both IDEs. For example, if you turn a preference on or off in one IDE, the same behavior occurs in the other IDE.

Restricting actions during the recording and playing back of tests with start application

You can restrict the recording of tests to capture only the actions that you perform on the start application. The playing back of such a test performs the recorded actions only on the instance of the start application associated with the test and reduces the overall playback time.

About this task

When you restrict the recording and playing back of tests that use a start application, the recording of a test does not capture any unnecessary click actions that you might perform on other active instances of the start application.

For information about how to add a start application to a test, refer to Related Links.

If you enable the **Limit Record/Playback to StartApp applications only** option, the following changes are applied to the recording and playing back of tests that use a start application:

- The recording monitor captures only the actions that you perform on the instance of the start application associated with the test during recording.
- The playing back of the test performs the verification points and data-driven commands only on the instance of the start application associated with the test.
- If you use the **Find the object** option to insert an object when an `ObjectNotFound` exception is displayed during playback, you can select only the missing objects from the instance of the start application associated with the test.

  Note: In Rational® Functional TesterHCL OneTest™ UI running on Linux, you cannot insert an object by using the **Find the object** option when an `ObjectNotFound` exception is displayed. Therefore, you must disable the **Limit Record/Playback to StartApp applications only** option and then play back the test.

- If you have a test that has `find()`methods, the playing back of the test performs the `find()`method based actions only on the instance of the start application associated with the test and reduces the overall playback time.

Note: The feature to restrict the recording and playing back of tests that use a start application does not apply to browsers during playback.

1. Go to **Window > Preferences > Functional Test** in Rational® Functional TesterHCL OneTest™ UI.
2. Select **Limit Record/Playback to StartApp applications only**.

   Important: If you remove the start application from the test, then the playback is not restricted to the start application associated with the test.

Results

You have restricted the recording of tests to capture only the actions that you perform on the start application.

Related reference

Start application dialog box on page 1837

Exception dialog box on page 1887

Related information

Handling exceptions during script playback on page 871

How to handle objectnotfound exception in Rational Functional Tester

Using a keyboard shortcut to record an application instance

If you want to capture only the actions that you perform on a specific application instance during the recording of a test script, you can use a keyboard shortcut to enable that application instance to be recorded. The recording monitor then captures only the actions that you perform on the application that you enabled by using the keyboard shortcut.

1. Go to **Window > Preferences > Functional Test** in Rational® Functional TesterHCL OneTest™ UI.
   The **Type StartApp key combination** field is enabled by default. The default keyboard shortcut to enable an application instance to be recorded is left **Ctrl**+ right **Shift**. You can change the default keyboard shortcut by completing the following steps:
   a. Clear the **Use default** check box for the **Type StartApp key combination** field.
   b. Enter a keyboard shortcut in the **Type StartApp key combination** field.

      Important:

      You can use only the following keyboard shortcuts to enable an application instance to be recorded when you record a test script:

- Left **Ctrl** + right **Shift**
- Left **Ctrl** + any alphabet key

c. Click **Apply**.

You have modified the default keyboard shortcut. You must use this keyboard shortcut to enable an application instance to be recorded.

2. Select the **Functional Test** perspective.
3. Click the **Record a Functional Test Script** icon from the Rational® Functional TesterHCL OneTest™ UI toolbar.
   The **Record an IBM Rational Functional Tester scriptRecord an HCL OneTest UI script** dialog box is displayed.
4. Complete the fields to record a test script and click **Finish**.
   The recording monitor window opens.
5. Click the instance of the application that you want to record and then press the keyboard shortcut that you created.
   The application is enabled to be recorded, and the following message is displayed in the recording monitor:

   ```
   <application name> application added successfully
   ```

   The recording of the test script captures only the actions that you perform on the enabled application instance. All actions that you perform on any other application instances are not captured.

What to do next

You can perform the actions for the enabled application instance and complete the recording.

Related reference

The Rational Functional TesterHCL OneTest UI toolbar on page 1777

Record a New Functional Test Script dialog box on page 1800

Changing the verification point and object map colors

You can change the color of information displayed in the Verification Point Editor, Verification Point Comparator, and Object Map. This feature is useful for enhancing the accessibility for people with disabilities.

1. In the product menu, click **Window > Preferences** to display the Preferences dialog box.
2. In the Preferences dialog box, expand **Functional Test** in the left pane.
3. Click **Colors** to display the Colors page on page 830.
4. Select one of the following tabs:
   Choose from:
   - Verification Point Editor
   - Verification Point Comparator
   - Object Map Editor

5. Select a display item in the list.
6. Change the color for the display item:
    a. Click the **Color** button.
    b. Click a color in the **Basic colors** selection palette.
    c. Click **OK**.
7. Click **Apply** to save the new color setting and continue changing colors or click **OK** to save the new color setting and close the Preferences dialog box.

Changing the syntax of Verification Point commands

During recording, a verification point captures information about a specified GUI component, for example, its size or its position. Rational® Functional TesterHCL OneTest™ UI records a statement in the test script for each verification point.

About this task

Verification point commands appear in the script with the name that you assign to the verification point when you record the script.

The new syntax for the verification point command is:`ObjectName.performTest(VPName);`

For example, if we capture a verification point called ChkBox_State on a check box, say CheckBox1 it is recorded as `CheckBox1().performTest(ChkBox_State());`

The verification point with this syntax stores only the information for the expected value and does not associated it with a specific GUI object. This provides the flexibility to reuse the verification point on similar objects.

For example: Use the same ChkBox_State verification point on another check box. You can use the expected value information stored in the ChkBox_State verification point to validate the actual value obtained from CheckBox2 control. You can use the statement `CheckBox2().performTest(ChkBox_State());`

In earlier versions of Rational® Functional TesterHCL OneTest™ UI this syntax was:`VP().performTest();`

For example, `CheckBox1_StateVP().performTest();` This verification point cannot be reused to validate the contents of another check box control. You cannot use the expected value stored in the CheckBox1_State verification point to validate the actual value obtained from the CheckBox2 control.

You can choose which one to use. By default the new syntax is enabled. If you want to use the old syntax, follow these steps:

1. Click **Window > Preferences**.
   Result
   The Preferences dialog box appears.
2. Click **Functional Test > Recorder** and clear the **Record Test Object relative Verification Point** option.
3. Click **Apply** and **OK** to close.
   Result
   The next time you record or insert a new verification point, it will show the old syntax in the test script.

Changing user interface fonts

You can change the font and point size of information displayed in dialog boxes for Rational® Functional TesterHCL OneTest™ UI and those that are part of the Eclipse IDE. For IDE dialog boxes and the Java™ Editor, you do this through the Workbench Preferences.

1. Click **Window > Preferences**, expand **Workbench** , and click **Colors and Fonts**.
2. In the **Colors and Fonts** list, select the type of text you want to change. On the **Colors and Fonts** preference page, you can change the banner font, header font, and text font, one at a time.
3. To use system fonts, click **Use System Font**. You must use this option for DBCS languages.

   To use a different font, click **Change**. In the Font dialog box, select your font and click **OK**.
4. To change the font in the Java™ Editor, in the **Colors and Fonts** list, take the following steps:
   Choose from:
   - Expand **Java**.
   - Click **Editor Text Font (defaults to Text Font)**.
   - Click **Restore Defaults**.

   Result
   The Java™ Editor picks up the font set in the Workbench **Font** preference page because the Java™ Editor uses that preference by default.
5. Click **OK**.
   The new font does not take effect immediately in the dialog box that you set it in (Application Configuration Tool or Enable Environments dialog box).

Rational® Functional TesterHCL OneTest™ UI General page

You use the Rational® Functional TesterHCL OneTest™ UI General page to set all product time options. These options are useful to accommodate different computer speeds.

The General page has the following controls:

Limit Record/Playback to StartApp applications only: Select this option to limit the recording and playing back to StartApp application only.

Automatic enablement: Automatic enablement is activated by default. Deselect the checkbox if you want to statically enable each test environment. This is useful for improving the performance of tests.

Multiply all time options by: Enter any real number by which you want to multiply all Rational® Functional TesterHCL OneTest™ UI preferences or options that take an amount of time as an argument. For example, enter .5 to make all Rational® Functional TesterHCL OneTest™ UI time options twice as fast. This option affects all the following controls:

General Playback

Maximum time to attempt to find Test Object

Pause between attempts to find Test Object

Timeout used in waitForExistence( ) method

Retry timeout used in waitForExistence( ) loop

Delays

Delay before mouse up

Delay before mouse move

Delay before mouse down

Delay before key up

Delay when hover

Delay after top level window activates

Delay before key down

Delay before performing Test Object action

General Recorder

Delay before recording a mouse action

Delay before recording a keystroke

Use Default

Clear this check box to edit the value in the **Multiply all time options by** field. Select this check box to restore the default value.

Restore Defaults

Restore the default values on this page.

Apply

Save your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, click **Functional Test**.

To open: Click **Tools > Options**. In the left pane, click **IBM Rational Rational Functional Tester**.

Related information

Restricting the actions during recording and playback

Colors page

You use the tabs on the Colors page to specify color settings for the Verification Point Editor, Verification Point Comparator, and the Object Map Editor. This feature is also useful for enhancing the accessibility for people who have physical challenges, such as limited vision.

Verification point editor

This tab contains the following controls:

Table compare region background -- Indicates the color of columns, rows, and cells that are going to be compared in the verification point editor.

Color -- Displays the color currently in use for the selected user interface element. Click to display a color selection palette from which you can click the color you want to use.

Verification point comparator

This tab contains the following controls:

Difference foreground -- Indicates the color of the difference in verification points in the verification point comparator.

Tree difference foreground -- Indicates the color of difference in the tree verification point in the verification point comparator.

Tree left node only foreground -- Indicates the color of the expected or baseline tree node that does not appear in the actual verification point in the verification point comparator.

Tree right node only foreground -- Indicates the color of actual tree node that only exists in the actual verification point and not in the expected or baseline verification point in the verification point comparator.

Table compare region background -- Indicates the color of columns, rows, and cells that are going to be compared in the verification point comparator.

Color -- Displays the color currently in use for the selected user interface element. Click to display a color selection palette from which you can click the color you want to use.

Object map editor

This tab contains the following controls:

Owned object foreground -- Indicates the color of the Owned: flag in the test object map.

Matching object foreground -- Indicates the color of objects matching the search criteria.

Delete All Not Used objects foreground -- Indicates the color of test objects found when searching for those not having references in the scripts associated with the shared test object map.

Delete Object Script Read-Only foreground -- Indicates the color of a read-only test object in the Delete Test Object dialog box.

Color -- Displays the color currently in use for the selected user interface element. Click to display a color selection palette from which you can click the color you want to use.

To open: Click **Window > Preferences**, expand **Functional Test**, and click **Colors**.

Highlight page

You use the Highlight page to specify how you want Rational® Functional TesterHCL OneTest™ UI to emphasize test objects in applications-under-test when you select them in a test object map or in the Script Explorer. These settings also control how Rational® Functional TesterHCL OneTest™ UI highlights objects you select with the Verification Point and Action Wizard and the Insert a GUI Object into the Object Map dialog box.

You can also change these settings in the test object map by clicking **Preferences > Highlight** on the test object map menu.

The Highlight page has the following controls:

Color

Click to display a color selection palette from which you can select a color to use to indicate selected test objects. The button displays the color currently in use.

Border Width (in pixels)

Move the slider from **Thin** to **Wide** to set the width of the border around the selected object.

Flash Speed

Move the slider from **Slow** to **Fast** to set the rate at which the border around a selected object flashes when selected.

Display Time

Move the slider from **Short** to **Long** to set the length of time to highlight the border.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, expand **Functional Test** and click **Highlight**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Highlight**.

Logging and Tracing page

IBM® Rational® Functional TesterHCL OneTest™ UI allows you to collect the errors and warning messages into a log file (rft_log.txt). The logging and tracing functionality is controlled through certain configurable parameters such

as level, file size, directory. With IBM® Rational® Functional TesterHCL OneTest™ UI you can configure the 'level' parameter for individual components. If a log and trace level is not defined explicitly for a component, then it defaults to the overall setting defined in the general page. With IBM® Rational® Functional TesterHCL OneTest™ UI, you can set the preferences for logging and tracing through an user interface.

General page

The General page contains the fields listed below.

Note: These settings do not apply to messages logged for functional testing in Google Chrome browsers. Messages for Google Chrome are written into the chromeSupportDebug.txt file, which is saved in the directory specified in the **Log File Path** field.

Log Level

Select the log level from the list. With the log level you can control the amount and type of log information that must be generated in the log file. For example, if you select **INFORMATION** level, all the log messages that are classified as information, warning, error, fatal, and severe is generated in the log. To minimize the amount of logging, select **FATAL**. If you select **CONFIG**, all log messages are generated.

Log File Size (in KB)

Specify the maximum size of the log file in kilobytes. The default size is 2048 KB, and the minimum size of the log file is 1024 KB. If the size of the log file exceeds the specified limit, it is renamed to rft_log_<x>.txt (The size of the renamed file is approximately around the specified limit).

Maximum Number of Log files to retain

Specify the maximum number of log files that can be retained. For examples, if you say type, 5, then five recent log files are retained and the rest are cleared.

Log File Path

Type the log file path in the **Log File Path** field. The log file generated (rft_log.txt) is saved in this directory.

Note: Messages for Google Chrome are written into the chromeSupportDebug.txt file, which is saved in the directory specified in this field.

You must enable the tracing option, if you want to generate the trace files.

Enable Tracing

Select this check box to enable the generation of trace files.

Generate traces in Eclipse error log format

Select this check box to enable the generation of trace files in the Eclipse error log `.log` format. Trace files in the `.log` format can be viewed in the Eclipse Error Log view within Rational® Functional TesterHCL OneTest™ UI. Clear this check box to enable trace files to be generated in the `.txt` format.

Trace Level

Select the trace level from the list. With the trace level you can control the amount and type of trace information that must be generated in the trace file. The trace level gives you detailed debugging information for Rational® Functional TesterHCL OneTest™ UI and the application-under-test during recording and playback.

Trace File Size (in KB)

Specify the maximum size of the trace file in kilobytes. The default size is 2048 KB, and the minimum size of the trace file is 1024 KB. If the size of the trace file exceeds the specified limit, it is renamed to rft_trace_<x>.txt (The size of the renamed file is approximately around the specified limit).

Maximum Number of Trace files to retain

Specify the maximum number of trace files that can be retained. For examples, if you say type, 5, then five recent trace files are retained and the rest are cleared.

Trace File Path

Type the trace file path in the **Trace File Path** field. The trace file generated (rft_trace.txt or rft_trace.log, depending on your specification in the **Generate traces in Eclipse error log format** check box) is saved in this directory.

Log Components and Trace Components

The component level settings take precedence over the general settings. The components are not predefined. You can add and remove components by clicking the **Add** and **Remove** buttons.

Log Components

On the **Log Components** pages, you define the log level settings for each component.

Trace Components

On the **Trace Components** pages, you define the trace level settings for each component.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test** and click **Logging and Tracing**.

Memory trace components

The Memory Trace page contains settings which control the type of trace statements to be generated for Rational® Functional TesterHCL OneTest™ UI processes.

Memory Trace Settings

Click this option to enable memory trace settings.

Interval between dumps

Specify the intervals, in seconds, at which the Java heap size statistics are to be dumped into the memory trace.

Run Garbage Collector prior to dumping

Click this option to indicate that the Garbage Collector needs to run before the Java heap size statistics are dumped into the memory trace file.

Write the shared memory to the trace file

Click this option to enable writing the shared memory statistics to the trace file.

Logging page

You use the Logging page to set log and comparator options, such as preventing the script launch wizard from displaying on playback, displaying the log viewer after playback, and displaying a message before overwriting an existing log. You also use this page to indicate the type of log generated.

To access the logging page, click **Window > Preferences**. In the left pane, expand **Functional Test > Playback** and click **Logging**.

Note: For Microsoft Visual Studio, click **Tools > Options**. In the left pane, expand **Functional Test > Playback** and click **Logging**.

The Logging page contains the following options:

Don't show script launch wizard: When selected, prevents the script launch wizard from displaying each time you play back a script.

Display log viewer after script playback: When selected, this option displays the log after you play back a script. If the log type is HTML, the log opens in your default browser. If the log type is Text, the log opens in the Script Window of Rational® Functional TesterHCL OneTest™ UI. If the log type is XML, the log opens in your default browser.

Generally, the log file opens in the default browser that is associated with the html file extension in your computer. To view the html files in your desired browser, you can associate the html file extension with the specific browser. The file extension for different browsers are as follows:

- For Google Chrome, you must associate .html=ChromeHTML
- For Internet Explorer, you must associate .html=htmlfile
- For Firefox, you must associate .html=FirefoxHTML-308046B0AF4A39CB

Log screen snapshot for each action on the application: When selected, this option records a screen snapshot in the playback log against every action performed on the application.

Prompt before overwriting an existing log: When selected, this option prompts you before you overwrite a log.

Log the count of test objects created/unregistered at particular script line: When selected, this option logs these details:

- Number of objects created and unregistered at a specific script line
- Total number of objects created and unregistered per call script
- A cumulative number of test objects created and unregistered for the whole script during playback if Rational® Functional TesterHCL OneTest™ UI scripting methods have been used to return test objects.

Warning messages are also logged at the call script level and the main script level, if the number of test objects created exceeds the number of test objects unregistered, which would suggest the possibility of memory leaks during playback.

Log a screen snapshot when playback fails: When you select this option, it captures a screen snapshot at the time of the failure and stores it in the log. You must clear the check box to save storage space (172 KB per snapshot).

Log GUI actions on the application: When you select this option, it adds a detailed record of any GUI-related actions performed on the application (without a screen snapshot) to the playback log.

Log type: This option Indicates the type of log Rational® Functional TesterHCL OneTest™ UI generates to write results of script playback. The log types are as follows:

- **None**: Generates no log, if selected.
- **Text**: Displays the log in ASCII format in the Functional Test script window.
- **HTML**: Displays the log in HTML format in your default browser. The left pane in the HTML log contains three boxes: Failures, Warnings, and Verification Points. The list of items in each box help you navigate to a specific location in the log. You can select an item to quickly find important errors, warnings, and verification point results in the log. To do so, double-click an item in a list, and Rational® Functional TesterHCL OneTest™ UI scrolls to and displays the item in the log.
- **TPTP**: Displays a log using TPTP in the Functional Test script window.
- **XML**: Displays a log of XML data rendered in HTML format [using transformation and Cascaded Style Sheets] in your default browser.
- **Default**: Displays the unified report for the test scripts in the browser window. This is also the default option to generate result for Functional test scripts.
- **JSON**: Displays a log in JSON format in the Functional Test Script window. Each event in this log type is a separate JSON.



Note: The JSON log type is not supported in the integration of Rational® Functional TesterHCL OneTest™ UI with Visual Studio.

Use Default: Clear the check box to change the value in the **Log type** field. Select the check box to restore the default value.

Restore Defaults: Restores the default values on this page.

Apply: Saves your changes without closing the dialog box.

Operating System page

You use the Operating System page to indicate the Foreground Lock Timeout setting for Windows® 98/Me and Windows® 2000 systems.

This page contains the following controls:

Foreground Lock Timeout

An important option for **Windows 98/Me**, **Windows 2000**, or later that sets the amount of time (in milliseconds) after user input, during which the operating system does not allow applications to force themselves into the foreground. To play back scripts, you must change this setting to **0**. However, this is a persistent setting and affects desktop behavior.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open: Click **Windows > Preferences**. In the left pane expand **Functional Test** and click **Operating System**.

To open: Click **Tools > Options**. In the left pane expand **IBM Functional Test** and click **Operating System**.

To open: Click **Tools > Options**. In the left pane expand **HCL Functional Test** and click **Operating System**.

General Playback page

You use the General Playback page to set script playback options, such as the amount of time Rational® Functional TesterHCL OneTest™ UI looks for an object and waits before trying to find an object again. You can also elect to skip all verification points in the script.

The General Playback page has the following controls:



Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Show exception dialog

The exception dialog box is displayed if an exception occurs during playback.

Perform playback in interactive mode

To resolve common runtime situations dynamically.

Maximum time to attempt to find Test Object

The maximum amount of time, in seconds, that Rational® Functional TesterHCL OneTest™ UI attempts to find an object. See example on page 838.

Pause between attempts to find Test Object

Indicates, in seconds, how long Rational® Functional TesterHCL OneTest™ UI waits before trying to find an object again. See example on page 838.

Skip Verification Points

When selected, skips all verification points in the script.

Timeout used in waitForExistence( ) method

Indicates, in seconds, the maximum amount of additional time that Rational® Functional TesterHCL OneTest™ UI waits (after time specified in **Maximum time to attempt to find test object**) for an object. For example, this setting is useful when waiting for an application to open. The waitForExistence( ) method must be explicitly stated in the script.

Retry time used in waitForExistence( ) loop

Indicates, in seconds, the interval between attempts to find an object. If Rational® Functional TesterHCL OneTest™ UI does not find an object, it continues to try until the time specified in **Timeout used in waitForExistence( ) method** has expired.

Restore Defaults

Restores the default values on this page.

Apply

Saves the edits you made without closing the dialog box.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test** and click **Playback**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Playback** or **General.**

Related reference

Setting general playback preferences in test scripts on page 837

Setting general playback preferences in test scripts

You can set the general playback preferences in test scripts. The preference set in the script, if any, takes precedence over the one that is set through the **Preferences** dialog and it applies globally to all the scripts. Hence, ensure that the script includes a reset command that reverts the preference to its earlier setting.

The following table lists the code snippets to be inserted for each general playback preference:

Preference	Code snippet

Show exception dialog	`setOption(IOptionName.SHOW_EXCEPTION_DLG, true);`
	`// actions`
	`resetOption(IOptionName.SHOW_EXCEPTION_DLG);`

Preference	Code snippet

Perform playback in interactive mode	`setOption(IOptionName.PERFORM_PLAYBACK_IN_INTERACTIVE_MODE, true);`
	`// actions`
	`resetOption(IOptionName.PERFORM_PLAYBACK_IN_INTERACTIVE_MODE);`
	
	Note:
	This setting works only if IOptionName.SHOW_EXCEPTION_DLG is set to **true**.

Maximum time to attempt to find Test Object	`setOption(IOptionName.MAXIMUM_FIND_OBJECT_TIME, maximumtimeinseconds);`
	`// actions`
	`resetOption(IOptionName.MAXIMUM_FIND_OBJECT_TIME);`

Pause between attempts to find Test Object	`setOption(IOptionName.FIND_OBJECT_DELAY_BETWEEN_RETRIES,`
	` pausetimeinseconds);`
	`// actions`
	`resetOption(IOptionName.FIND_OBJECT_DELAY_BETWEEN_RETRIES);`

Skip Verification Points	`setOption(IOptionName.SUPRESS_VP_PERFORM_TEST, true);`
	`// actions`
	`resetOption(IOptionName.SUPRESS_VP_PERFORM_TEST);`

Timeout used in waitForExistence() method	`setOption(IOptionName.MAXIMUM_WAIT_FOR_EXISTENCE,`
	` maximumwaittimeinseconds);`
	`// actions`
	`resetOption(IOptionName.MAXIMUM_WAIT_FOR_EXISTENCE);`

Retry time used in waitForExistence() loop	`setOption(IOptionName.WAIT_FOR_EXISTENCE_DELAY_BETWEEN_RETRIES,`
	` delaytimeinseconds););`
	`// actions`
	`resetOption(IOptionName.WAIT_FOR_EXISTENCE_DELAY_BETWEEN_RETRIES);`

Related reference

General Playback page on page 836

Playback preferences example on page 838

Playback preferences example

This topic provides an example of the interaction between settings in the Playback Preferences page.

If Rational® Functional TesterHCL OneTest™ UI does not find an object, it will continue to try until the time specified in **Maximum time to attempt to find test object** has expired. **Pause between attempts to find test object** indicates the amount of time to pause between retries.

For example, if **Maximum time to attempt to find test object** is set to 30.0, and the **Pause between attempts to find test object** is set to 1.0, Rational® Functional TesterHCL OneTest™ UI will look for an object up to 30 times, pausing 1 second between tries.

Dynamic Find Enablement page

You can use the Dynamic Find Enablement page to enable the dynamic find feature for all functional test scripts run within the integrated development framework (IDE). With the dynamic find feature, Rational® Functional TesterHCL OneTest™ UI can search for and locate objects in the script whose hierarchical position has changed, to prevent playback failure.

This page has this control:

Enable script find if scoring find fails

Enables the dynamic find feature for all scripts within the IDE.

Note: When you enable the dynamic find feature on this page, the setting applies to all functional test scripts that are run in the IDE. You can disable the feature for individual scripts on the Select Log page.

Delays page

You use the Delays page to set delays during Functional Test script playback. These settings are useful to control the rate at which script commands are sent to the operating system.

The Delays page has the following controls:

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Delay before mouse up

Indicates, in seconds, the interval before sending a mouse-release event during playback.

Delay before mouse move

Indicates, in seconds, the interval before sending a mouse-move event during playback.

Delay before mouse down

Indicates, in seconds, the interval before sending a mouse-press event during playback.

Delay before performing Flex Test Object action

Indicates, in seconds, the wait before performing a Flex test object action during playback.

Delay before key up

Indicates, in seconds, the wait before sending a key-release event during playback.

Delay when hover

Indicates, in seconds, the duration of the wait for a Hover command, which takes no options.

Delay after top level window activates

Indicates, in seconds, the wait after making a new window active. This provides the application time to repaint the screen.

Delay before key down

Indicates, in seconds, the interval before sending a key-press event during playback.

Delay before performing Test Object action

Indicates, in seconds, the time the object waits before each UI action.

Restore Defaults

Restores the values on this page to customization file settings (if they exist) or to RATIONAL._FT.RFTCUST settings.

Apply

Saves your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, expand **Functional Test**, expand **Playback,** and click **Delays**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Delays**.

Playback Monitor page

You use the Playback Monitor page to specify whether to display the playback monitor during playback.

The Playback Monitor page has the following controls:

Show monitor during playback

Displays the Playback Monitor during playback.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open, click **Window > Preferences**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Monitor**.click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Monitor**.

ScriptAssure page--Advanced

You use the ScriptAssure(TM) Advanced page to set thresholds for recognition scores, which Rational® Functional TesterHCL OneTest™ UI uses when searching for objects during script playback.

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

The ScriptAssure™ Advanced page has the following controls:

Maximum acceptable recognition score -- Indicates the maximum score an object can have to be recognized as a candidate. Objects with higher recognition scores are not considered as matches until the time specified in **Maximum time to attempt to find Test Object** has elapsed.

Last chance recognition score -- If Rational® Functional TesterHCL OneTest™ UI does not find a suitable match after the time specified in **Maximum time to attempt to find Test Object** has elapsed, indicates the maximum acceptable score an object must have to be recognized as a candidate. Objects with higher recognition scores are not considered.

Ambiguous recognition scores difference threshold -- Writes an AmbiguousRecognitionException to the log if the scores of top candidates differ by less than the value specified in this field. If Rational® Functional TesterHCL OneTest™ UI sees two objects as the same, the difference between their scores must be at least this value to prefer one object. You can override the exception by using an event handler in the script.

Warn if accepted score is greater than -- Writes a warning to the log if Rational® Functional TesterHCL OneTest™ UI accepts a candidate whose score is greater than or equal to the value in this field.

Standard -- Displays the Standard ScriptAssure(TM) preferences page, which enables you to use a slider to set the tolerance level from **Tolerant** to **Strict** for recognition levels and from **None** to **High** for warning levels.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves the edits you made without closing the dialog box.

Changes you make in this page are reflected in the ScriptAssure(TM) Page-Standard.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**. Click **Advanced**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**. Click **Advanced**.

ScriptAssure page-standard

During playback, Rational® Functional TesterHCL OneTest™ UI compares objects in the application-under-test with recognition properties in the test object map. You use the ScriptAssure(TM) Standard page to control object-matching sensitivity during playback. This feature enables you to successfully play back scripts when the application-under-test has been updated.

The ScriptAssure(TM) Standard page has the following controls:

Recognition Level -- Controls the level of recognition when identifying objects during script playback. To decrease tolerance for differences between the object in the application-under-test and the recognition properties, move the slider toward **Strict**. To increase the tolerance for differences, move the slider toward **Tolerant**.

- The maximum **Strict** setting indicates that objects must be an almost exact match. If only one important recognition property is wrong, Rational® Functional TesterHCL OneTest™ UI recognizes the object as a match after exhausting all other possibilities. An object with more than one wrong recognition property is not a match.
- The maximum **Tolerant** setting indicates that Rational® Functional TesterHCL OneTest™ UI selects an object with somewhat similar properties immediately.
- The default setting allows two important recognition properties to be wrong but still is a match if all other possibilities are exhausted. An object with more than two wrong recognition properties is not a match.

Warning Level -- Specifies when to be warned about differences between the object and the recognition properties. To increase the number of warnings, move the slider toward **High**. To decrease the number of warnings, move the slider toward **None**.

- The maximum **High** setting writes a warning to the test log of almost any difference. (Functional Test does not issue a warning when the difference is the browser.)
- The maximum **None** setting omits warnings to the test log of differences.
- With the default setting, Rational® Functional TesterHCL OneTest™ UI writes a warning to the test log whenever it finds a test object after the maximum time has elapsed during playback.

Advanced -- Displays the Advanced ScriptAssure Preferences page, which enables advanced users to set thresholds for recognition scores.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

Changes you make on this page are reflected in the ScriptAssure(TM) Page-Advanced.

To open: Click **Window > Preferences Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**.

Enabling the unexpected window handling feature

You can enable the unexpected window handling feature to ensure that scripts playback smoothly. When you enable this feature, unexpected windows that open during script playback are handled according to the configuration in the Configure Handling of Unexpected Windows dialog box.

Before you begin
Ensure that each unexpected window in the test domain has been configured in the Configure Handling of Unexpected Windows dialog box. By configuring all unexpected windows, the configured action is performed when a window opens unexpectedly during script playback.

1. In the product menu, click **Window > Preferences** to open the Preferences dialog box.
2. Expand **Functional Test**, and then click **Playback**.
3. Click **Unexpected Windows**.

4. Select the **Enable handling of unexpected windows** check box.

Note: When you enable the unexpected window handling feature in the Preferences dialog box, it applies to all scripts in the Integrated Development Environment. You can override this preference for an individual script in the Select Log page, when you run the script.

General Recorder page

You use the General Recorder page to indicate options for recording Functional Test scripts, such as excluding an executable from being recorded and setting the delay before recording a mouse action or a keystroke. You can also select or clear the option to open the test object map if there is a new test object in the application.

This page has the following controls:

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Record Test Object relative Verification Point -- When selected, the test object details are not recorded while inserting the verification points.

Maximum identifier length-- Option to control the maximum number of characters used in a Test Object identifier in the script.

Processes excluded from testing -- Enter the executable name of the process that you do not want to record. Separate multiple processes with commas.

Note: Only processes that are dynamically enabled must be added in this field.

Delay before recording a mouse action -- Sets the delay from the end of a mouse action to the command that appears in the recording monitor. A shorter delay may limit the quality of recording of actions that cause state changes. A value of 0.0 causes a delay until the beginning of the next action. If the value is lower than the double-click interval, Rational® Functional TesterHCL OneTest™ UI uses the double-click interval.

Delay before recording a keystroke -- Sets the delay from the last keystroke to the inputKeys command that appears in the recording monitor. A value of 0.0 causes a delay until the beginning of the next action.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

Bring up object map if there is new test object -- When selected, opens the test object map if a test object in the application is not currently in the map.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test** and click **Recorder**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Recorder** or **General**.

Recorder Monitor page

You use the Recorder Monitor page to change settings in the Recording monitor, such as displaying the recorder toolbar or the Recorder Monitor, including a timestamp for messages, and selecting the types of messages you want to display and their colors.

The Monitor page has the following controls:

Display recorder toolbar only -- Displays the recorder toolbar or the full Recorder Monitor window.

Include time stamp in the message -- Includes a timestamp, of the format *hh:mm:ss*, for each entry in the Recorder Monitor.

Error message color -- Indicates the color of errors in the Monitor. Double-click the color to change it.

Warning message color -- Indicates the color of warnings in the Monitor. Double-click the color to change it.

Information message color -- Indicates the color of information messages in the Monitor. Double-click the color to change it.

Select message types to display -- Enables you to include or omit any three message types that appear in the Monitor:

- Error
- Error, Warning
- Error, Warning, Information

You can add a time to messages in the Record Monitor, and indicate the types of and colors used for the messages. To do so, click the **Message Preferences** button  on the Record Monitor toolbar while recording.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

Mapping keyboard shortcut keys

You can map keyboard shortcut keys in Rational® Functional TesterHCL OneTest™ UI by changing the assignment of a shortcut key to a command. You can change the shortcut key for menu items and toolbar buttons.

1. Click **Window > Preferences**, expand **Workbench**, and click **Keys**.
   Result

   The Keys preference page appears.
2. On the **Keyboard Shortcuts** tab, under **Command**, take the following steps:

a. In the **Category** box, select the type of menu command for the shortcut key you want to change. For example, **Script**.
b. In the **Name** list, select a menu item name. For example, **Run**.
c. In the **Assignments** box, click **In Windows** or **In Dialogs and Windows**.

3. Under **Key Sequence**, take the following steps:

a. In the **Name** box, press the keys you want to assign to menu item. For example, Shift+Alt+F12.

Tip: To add keys to your key sequence, click the **Insert Special Key** arrow at the far right end of the **Name** box, and select a key from the pop-up menu. You can select **Backspace**, **Tab**, and **Shift+Tab**. For example, you can create a key sequence of F12, Backspace, Tab.

b. In the **When** box, select **In Windows** or **In Dialogs and Windows**.

4. Click **Add**.
5. Click **OK**.

Note: The new keyboard shortcut key does not take effect immediately for the menu item. After you close the Preferences dialog box, the change will take effect.

Customizing keyboard shortcuts in the Visual Studio IDE

You can customize the appearance and behavior of Rational® Functional TesterHCL OneTest™ UI to enhance accessibility for people with disabilities.

Keyboard shortcuts

You can use the keyboard shortcut keys to perform tasks in Rational® Functional TesterHCL OneTest™ UI.

Keyboard shortcut keys include:

- Underlined letter in a dialog box field label
- Underlined letter in a menu command
- Extended accelerator key, such as ALT, CTRL, and SHIFT
- Function keys F1 through F12

Dialog boxes

You can refer to the dialog boxes in Rational® Functional TesterHCL OneTest™ UI to find the keyboard shortcuts. When you open a dialog box, press the **ALT** key to display the underlined letter in a field label. Most field labels show the letter of the keyboard shortcut underlined. For example, **D** is the shortcut key for the **Devices** field label in the

Connect to Device dialog box. In Rational® Functional TesterHCL OneTest™ UI, press the **ALT** key, and the shortcut key to execute a command. For example, open the Connect to Device dialog box, press **ALT** to display the keyboard shortcuts in the dialog box, and press **ALT+D** to move to the **Devices** field in the Connect to Device dialog box.

Menu commands

You can refer to the menus in Rational® Functional TesterHCL OneTest™ UI to find the keyboard shortcuts. Most menu items show the equivalent keyboard shortcut to the right of the menu item or the letter of the keyboard shortcut underlined. For example, **F** is the shortcut key for the **File** menu. In Rational® Functional TesterHCL OneTest™ UI, press the **ALT** key, and then press the shortcut key to execute a command. For example, press **ALT+F** to open the **File** menu on the menu bar.

The following table describes the keyboard shortcuts:

Command Name	Key Sequence	Function
File Menu		
File.NewProject	ALT+F, N, P or CTRL+ SHIFT+ N	Create a new project.
File.File	ALT+F, N, F or CTRL+ N	Create a new file.
File.NewBlankSolution	ALT+F, N, B	Create a new blank solution.
File.NewProject	ALT+F, N, J	Create a new Functional Test project.
File.AddScriptUsingRecorder	ALT+F, N, U	Record a Functional Test script.
File.AddEmptyScript	ALT+F, N, E	Create an empty Functional Test script.
File.AddTestObjectMap	ALT+F, N, C	Create a test object map.
File.AddTestdataset	ALT+F, N, T	Create a test dataset.
File.OpenProject	ALT+F, O, P or CTRL+ SHIFT+ O	Open a project.
File.OpenProjectFromWeb	ALT+F, O, W	Open a project from the web.
File.OpenFile	ALT+F, O, F or CTRL+ O	Open a file.
File.FileFromWeb	ALT+F, O, E	Open a file from the web.
File.Convert	ALT+F, O, C	Convert a Java™ Language project to an equivalent Visual C# .NET project or convert a Visual Basic 6.0 project to an equivalent Visual Basic .NET project.
A command name is not assigned to the Close menu item.	ALT+F, C	Closes the script currently displayed in the Code Editor.
File.AddNewItem	ALT+F, W or CTRL+ SHIFT+ A	Add a new item to the currently selected project.
File.AddExistingItem	ALT+F, G or SHIFT+ALT+A	Add individual or multiple files to your project.

Command Name	Key Sequence	Function
File.NewProject	ALT+F, D, N	Add a new project.
File.OpenProject	ALT+F, D, E	Add an existing project.
File.OpenProjectFromWeb	ALT+F, D, F	Add an existing project from the web.
File.OpenProject	ALT+F, E	Open an existing solution (.sln) file.
	ALT+F, T	Close the current solution.
File.SaveSelectedItems	ALT+F, S or CTRL+ S	Save any changes you made to the script currently displayed in the Code Editor.
File.SaveSelectedItemsAs	ALT+F, A	Save the current file under another name and location.
File.SaveAll	ALT+F, L or CTRL+ SHIFT+ S	Save all the scripts and files currently open.
File.PageSetup	ALT+F, U	Display the Page Setup dialog box.
File.Print	ALT+F, P or CTRL +P	Display the Print dialog box.
A command name is not assigned to the recently used project items.	ALT+F , F , N	Display the names of project items you have most recently used.
A command name is not assigned to the recently used project names.	ALT+F , J , N	Display the names of projects you have most recently used.
A command name is not assigned to the Exit menu item.	ALT+F , X	Close Rational® Functional TesterHCL OneTest™ UI.
Edit Menu		
Edit.Undo	ALT+E , U or CTRL +Z	Cancel the last edit you made, if possible.
Edit.Redo	ALT+E , R or CTRL +Y	Do the action of the last Undo action again.
Edit.Cut	ALT+E , T or CTRL +X	Delete the selected characters and put them on the Clipboard.
Edit.Copy	ALT+E , C or CTRL +C	Copy the selected characters to the Clipboard.
Edit.Paste	ALT+E , P or CTRL +V	Insert at the cursor any characters that were previously cut or copied to the Clipboard.
Edit.CycleClipboardRing	ALT+E , Y or CTRL +SHIFT +V	Cycle through the contents of the Clipboard Ring tab.
Edit.Delete	ALT+E , D or Del	Remove the selected characters on the Code Editor or the selected item in the Solution Explorer.

Command Name	Key Sequence	Function
Edit.SelectAll	ALT+E , A or CTRL +A	Select all the characters in the currently displayed script or file.
Edit.Find	ALT+E , F , F or CTRL +F	Search for text in the current document or window.
Edit.Replace	ALT+E , F , P or CTRL +H	Search for and substitute text in the current document or window.
Edit.FindInFiles	ALT+E , F , I or CTRL +SHIFT +F	Search for text in the current document or window as well as within multiple documents.
Edit.ReplaceInFiles	ALT+E , F , S or CTRL +SHIFT +H	Search for and substitute text in the current document or window as well as within multiple documents.
Edit.FindSymbol	ALT+E , F , Y or ALT +F12	Search for a symbol in the current document or window.
Edit.GoTo	ALT+E , G or CTRL +G	Navigate to a specific line number in the Code Editor.
	ALT+E , X	Insert a file into a project as text.
Edit.FormatDocument	ALT+E, V, A or CTRL+K, CTRL +D	Format document with indents and space.
Edit.Format.Selection	ALT+E, V, F or CTRL+K, CTRL+F	Format selected text in the Code Editor.
Edit.MakeUppercase	ALT+E, V, U or CTRL+ SHIFT+ U	Make text uppercase.
Edit.MakeLowercase	ALT+E, V, L or CTRL+ U	Make text lowercase.
Edit.DeleteHorizontalWhitespace	ALT+E, V, H or CTRL+ K, CTRL+ \	Collapse white space in the selection or delete white space adjacent to the cursor if there is no selection.
Edit.ViewWhitespace	ALT+E, V, W or CTRL+ R, CTRL + W	Show or hide spaces and tab marks.
Edit.ToggleWordWrap	ALT+E, V, R or CTRL+ R, CTRL+ R	Word wrap.
Edit.IncrementalSearch	ALT+E, V, S or CTRL+ I	Perform an incremental search.
Edit.CommentSelection	ALT+E, V, M or CTRL+ K, CTRL + C	Add a comment to the current line of code.
Edit.UncommentSelection	ALT+E, V, E or CTRL+ K, CTRL+ U	Remove the comment from the current line of code.
Edit.ToggleBookmark	ALT+E, K, T or CTRL+ K, CTRL+ K	Set or remove a bookmark at the current line.

Command Name	Key Sequence	Function
Edit.NextBookmark	ALT+E, K, N or CTRL+ K, CTRL + N	Navigate to the next bookmark.
Edit.PreviousBookmark	ALT+E, K, P or CTRL+ K, CTRL+ P	Navigate to the previous bookmark.
Edit.ClearBookmarks	ALT+E, K, C or CTRL+ K, CTRL + L	Remove all unnamed bookmarks.
Edit.ToggleTaskListShortcut	ALT+E, K, H or CTRL+ K, CTRL + H	Add a task list shortcut.
Edit.HideSelection	ALT+E, N, H or CTRL+ M, CTRL + H	Hide selected text in the outline.
Edit.ToggleOutliningExpansion	ALT+E, N, T or CTRL+M, CTRL +M	Toggle between expanding and collapsing the selected text in the outline.
Edit.ToggleAllOutlining	ALT+E, N, L or CTRL+M, CTRL +L	Toggle between hiding and displaying the selected hidden text in the outline.
Edit.StopOutlining	ALT+E, N, P or CTRL+M, CTRL +P	Remove all outlining.
Edit.StopHidingCurrent	ALT+E, N, U or CTRL+M, CTRL +U	Remove the outlining for the selected text.
Edit.CollapseToDefinitions	ALT+E, N, O or CTRL+M, CTRL +O	Collapse the outline to display only definitions.
Edit.ListMembers	ALT+E, I, B or CTRL+J	List members of the current class for statement completion when you edit code.
Edit.ParameterInfo	ALT+E, I, O or CTRL+SHIFT +Space	Display a tool tip that contains information for the current parameter.
Edit.QuickInfo	ALT+E, I, Q or CTRL+K, CTRL+ I	Display Quick Info.
Edit.CompleteWord	ALT+E, I, W or ALT+Right Arrow	Display Word Completion.
View Menu	The View menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Script Explorer, Test dataset, and Solution Explorer menu items. For more information, see the Visual Studio Default	

Command Name	Key Sequence	Function
	Settings Shortcut Keys online Help.	
View.ScriptExplorer	ALT+V, X	Display the Script Explorer with a list of the script helper, helper superclass or helper base class, test dataset, verification points, and test objects for the current script.
View.Testdataset	ALT+V, D	Display the test dataset editor.
View.SolutionExplorer	ALT+V, P	Display the Solution Explorer, which provides an organized view of your projects and their files as well as access to the commands that pertain to them.
Project Menu		
File.AddScriptUsingRecorder	ALT+P, Enter	Record a Functional Test script.
File.AddEmptyScript	ALT+P, E	Create an empty Functional Test script.
File.AddTestObjectMap	ALT+P, C	Create a test object map.
File.AddTestdataset	ALT+P, T	Create a test dataset.
Project.AddHelperBaseClass	ALT+P, H	Create a new script helper base class.
Project.AddWindowsForm	ALT+P, F	Create a Visual Basic windows form
Project.AddInheritedForm	ALT+P, I	Create a Visual Basic windows form based on an existing windows form.
Project.AddUserControl	ALT+P, U, Enter	Create a Visual Basic user control
Project.AddInheritedControl	ALT+P, T	Create a Visual Basic user control based on an existing user control.
Project.AddComponent	ALT+P, N	Create a Visual Basic component
Project.AddModule	ALT+P, M	Create a Visual Basic module
Project.AddClass	ALT+P, C, C, Enter	Create a Visual Basic empty class declaration.
File.AddNewItem	ALT+P, W or CTRL+SHIFT+A	Display the Add New Item dialog box to add a form for Windows® applications.
File.AddExistingItem	ALT+P, G or SHIFT+ALT+A	Display the Add Existing Item dialog box to select individual or multiple files to add to your project.
Project.AddReference	ALT+P, R	Display the Add Reference dialog box to list the various types of components and projects you can browse.

Command Name	Key Sequence	Function
Build Menu	The Build menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI. For more information, see the Visual Studio Default Settings Shortcut Keys online Help.	
Debug Menu	The Debug menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI. For more information, see the Visual Studio Default Settings Shortcut Keys online Help.	
Script Menu		
Debug.StartWithoutDebugging	ALT+S, R or CTRL+F5	Compile and run the Rational® Test script
Debug.Start	ALT+S, D or F5	Compile and debug the Rational® Test script
File.AddScriptUsingRecorder	ALT+S, U	Display the Add New Item dialog box to enter information about the new script and start recording.
Script.InsertRecording	ALT+S, N	Start recording at the cursor location in the current script and start applications, insert verification points, and add script support functions.
Script.OpenTestObjectMap	ALT+S, O	Open the test object map associated with the current script.
Script.UpdateScriptHelper	ALT+S, H	Update the script's helper file (*ScriptHelper.vb) to reflect changes made to the template for the selected test asset.
Script.InsertTestObject	ALT+S, T	Display the Insert a GUI Object into the Object Map wizard to add objects to the test object map.
Script.InsertVerificationPoint	ALT+S, V	Display the Verification Point and Action Wizard to insert a verification point in the current script at the cursor location.

Command Name	Key Sequence	Function
Script.InsertDataDrivenCommands	ALT+S, D	Display the Data Drive Actions page of the dataset Population Wizard to select the objects in an application-under-test to data-drive an application.
Script.FindLiteralsandReplacewithdatasetReference	ALT+S, F	Replace literal values with a dataset reference in a test script to add realistic data to an existing test script.
Configure Menu		
Configure.ConfigureApplicationsforTesting	ALT+U, C	Display the Application Configuration Tool to add and edit configuration information -- such as name, path, and other information that Rational® Functional TesterHCL OneTest™ UI uses to start and run the application -- for the Java™ and HTML applications you want to test.
Configure.EnableEnvironmentsforTesting	ALT+U, E	Display the Enable Java™ Environments (JRE)/ Web Browsers/Eclipse Platforms for Testing dialog box to enable Java™ environments, browsers, and Eclipse platforms and to configure them.
Tools Menu	The Tools menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Test Object Inspector, Synchronize Solutions with Editor, and Options menu items. For more information, see the Visual Studio Default Settings Shortcut Keys online Help.	
Tools.TestObjectInspector	ALT+T, I	Display the Test Object Inspector window to dynamically view properties of selected test objects visible in the running application and display information about those objects, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information
Tools.SynchronizeSolutionwithEditor	ALT+T, W	Synchronize a solution with the Code Editor.

Command Name	Key Sequence	Function
Tools.Options	ALT+T, O	Customize Rational® Functional TesterHCL OneTest™ UI by setting preferences for Rational® Functional TesterHCL OneTest™ UI, Highlight, Operating System, Playback, Recorder, Development Environment, and Advanced Development Environment.
Window Menu	The Windows® menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the recent scripts and Windows® menu items. For more information, see the Visual Studio Default Settings Shortcut Keys online Help.	
A command name is not assigned to the recently used script names.	ALT+W , N	Display the names of scripts you have most recently used.
Window.Windows	ALT+W, W	Display, close, and arrange documents that are open in editors and designers.
Help Menu		
Help.DynamicHelp	ALT+H, H or CTRL+F1	Display links to topics related to your current work in the Dynamic Help window. The links displayed are based on the portion of the integrated development environment that currently has focus.
Help.Contents	ALT+H, C or CTRL+ALT+F1	Display the table of contents in the Contents window.
Help.Index	ALT+H, I or CTRL+ALT+F2	Display a list of keywords, arranged alphabetically, in the Index window.
Help.Search	ALT+H, S or CTRL+ALT+F3	Display the Search window to locate all Library topics for a particular word or phrase.
Help.Indexresults	ALT+H, X or SHIFT+ALT+F2	Display the Index Results window to list the topics associated with the keyword you selected in the Index window. The Index Results window is automatically displayed if the keyword you selected has more than one topic associated with it.

Command Name	Key Sequence	Function
Help.Searchresults	ALT+H, R or SHIFT+ALT+F3	Display the Search Results window to view the topics that include matches to the text you entered in the Search window.
Help.Edit.Filters	ALT+H, F	Display the Edit Help Filters window to change edit filter definitions.
Help.ShowStartPage	ALT+H, O	Display the Start Page to easily access recent projects and quickly create a new project.
Help.CheckforUpdates	ALT+H, K	Check for available Service Releases.
	ALT+H, T	Provide links to get support information based on the software and associated Help collections that were provided to your computer during software installation.
Help.HelponHelp	ALT+H, E	Provide information for using the online Help collections.
Help.About	ALT+H, A	Display information about the current version of Microsoft® Visual Basic .NET you are running.

Related Topics:

Mapping Keyboard Shortcut Keys on page 854

Customizing Rational® Functional TesterHCL OneTest™ UI for Accessibility on page 845

Colors Page

Changing User Interface Fonts

Mapping keyboard shortcut keys

You can map keyboard shortcut keys in Rational® Functional TesterHCL OneTest™ UI by assigning a shortcut key to a command. You can change the shortcut key for menu items.

About this task

Before you can map a keyboard shortcut key, you must know the internal command name for the menu item. To find the internal command name, see Keyboard Shortcuts on page 845 and jot down the internal command name for the menu item for later use in mapping a keyboard shortcut key.

To map a keyboard shortcut key:

1. Click **Tools > Options > Environment > Keyboard**.
2. In the **Show Commands Containing** field, type the internal command name for the toolbar button or menu item. For example, File.AddScriptUsingRecord.
3. In the **Press shortcut key(s)** field, type the shortcut key combination you want to assign. For example, press Shift+Backspace.
4. Click **Assign**.
5. Click **Yes** to save the keyboard scheme.
6. Type a name for the keyboard scheme. For example, Rational® Functional TesterHCL OneTest™ UI. Click **OK**.
7. Click **OK**.

Note: The new keyboard shortcut key does not take effect immediately for the toolbar button or menu item. After you close the Preferences dialog box, the change will take effect.

Simplified Scripting preference page

Use this page to indicate whether the simplified scripts feature must be enabled while creating the test scripts.

Simplified test scripts are displayed as English statements in the **Script** editor that are easy to understand and edit. With this feature enabled, you can also view the Java code of the test script in the **Script** editor. The simplified script feature is enabled by default.

Enable Simplified Scripting

Select the check box to enable the simplified scripting feature. If you clear this check box you can view only the Java code of the test script.

If you have enabled simplified scripting, and want to use Java scripting to record an individual script, you can override this setting by selecting **Java Scripting** from the **Select Mode** list in the **Record a Functional Test Script** dialog box when you record the script.

Application Visuals preference page

Use this page to specify whether to capture the application visuals of the test application while recording the test scripts. You can also specify whether to enable the verification points or data-driven commands that are featured in the application visuals so that you can insert them in the script using the application visuals without opening the test application.

Enable capturing Application Visuals

Select the check box to capture the application visuals while recording the test scripts.

Insert Data Driven Commands

Select the check box to enable the option to insert the data-driven commands into the script from the application visual.

Show Verification Point Dialog

Select the check box to use the verification point wizard while inserting the data verification points in the script using the application visual.

Enable capturing of verification on test data

Select the check box to enable the option to insert a verification point in the test script using the application visuals displayed in the **Application View**.

Simplified Script Editor preference page

Use this page to specify the preferences for simplified script editor operations such as undo, redo and undo history operations.

Undo history limit

Specify the limit for storing the history of delete actions that you perform in a script for performing the undo and redo operations.

Confirm all undo operations

Select the check box to enable the option to confirm the undo operations that you perform in the simplified script editor.

Confirm all delete operations

Select the check box to enable the option to confirm the delete operations that you perform in the simplified script editor.

Webserver Configuration page

You can use the Webserver Configuration page to change the default web server port for communication between Google Chrome and Rational® Functional TesterHCL OneTest™ UI, and to enable logging of messages.

The web server port is used when you enable the Google Chrome browser manually in the **Enable Environments for Testing** dialog box. By default, the port 9100 is set for the web server. If this port is already in use, change it and specify an available port.

The Webserver Configuration page contains the following controls:

Webserver Port

This field identifies the web server port for communication between Google Chrome and Rational® Functional TesterHCL OneTest™ UI. If the default port 9100 is already in use, change it and specify an available port.



Note: If you change the port in the Webserver Configuration page, ensure that you also make the same change in the Options for the IBM® Rational® Functional TesterHCL OneTest™ UI for

Google Chrome™ extension. For instructions to do this, see Changing the web server port for communication with Google Chrome on page 789.

Enable Webserver Logging

Select this check box to enable logging of messages for the JavaScript classes in the Google Chrome browser. The messages are logged in the `chromeSupportDebug.txt` file that is found in the log file path specified in the Logging and Tracing page.

Note: After setting your configuration preferences in this page, restart Rational® Functional TesterHCL OneTest™ UI for your changes to take effect.

Workbench Preferences page

The Workbench Preferences page enables you to indicate how you want the Workbench to behave while playing back, recording, and debugging Functional Test scripts.

The Workbench page has the following controls:

Workbench state during run -- Enables you to indicate how you want the Workbench to display while playing back scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during playback.
- **Minimized and restored on playback termination (Default)** -- Reduces the Workbench to a button on the taskbar during playback and restores it when playback finishes.
- **Hidden** -- Hides the Workbench during playback and restores it when playback finishes.
- **Leave in current state** -- Does not change the Workbench during playback.

Workbench state during recording -- Enables you to indicate how you want the Workbench to display while recording scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during recording.
- **Minimized and restored when recording finished (Default)** -- Reduces the Workbench to a button on the taskbar during recording and restores it after recording stops.
- **Hidden** -- Hides the Workbench during recording and restores it after recording stops.
- **Leave in current state** -- Does not change the Workbench during recording.

Workbench state during debug -- Enables you to indicate how you want the Workbench to display while debugging scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during debugging.
- **Minimized and restored on playback termination** -- Reduces the Workbench to a button on the taskbar during debugging and restores it after debugging stops.
- **Hidden** -- Hides the Workbench during debugging and restores it after debugging stops.
- **Leave in current state (Default)** -- Does not change the Workbench during debugging.

Restore Defaults -- Restores all the default values on this page.

Apply -- Saves the edits you made without closing the dialog box.

To open: Click **Window > Preferences**. In the left pane expand **Functional Test** and click **Workbench**.

Workbench Advanced Preferences

This Advanced page enables you to set advanced Workbench preferences for IBM® Rational® Functional TesterHCL OneTest™ UI, such as switching to the Test Debug perspective rather than the Functional Test perspective when debugging or turning on or off.

The Advanced page has the following controls:

Switch to Test Debug Perspective when debugging -- When selected, IBM® Rational® Functional TesterHCL OneTest™ UI switches to the Test Debug perspective when you select **Script > Debug**. When cleared, IBM® Rational® Functional TesterHCL OneTest™ UI continues to display the Functional Test perspective while the script runs in debug mode.

Restore Defaults -- Restores all the default values on this page.

Apply -- Saves the edits you made without closing the Preferences dialog box.

To open: Click **Window > Preferences**. In the left pane expand **Product Name > Workbench**, and click **Advanced**.

Managing functional test projects

A functional test project stores application test assets such as scripts, object maps, verification point baseline files, and script templates. You must create a functional test project before you can record scripts.

Creating a test project

A test project stores application test assets such as scripts, object maps, verification point baseline files, and script templates. You must create a test project before you can record scripts. You must create a new test project or connect to an existing test project before you record a new script.

1. From the product menu, click **File > New > Functional Test Project**.
2. In the **Project name** field, type the name of the new project.

   Names for functional test projects cannot contain the following characters: \ / : * ? " < > | ( ) or a space.

3. In the **Project location** field, type the data path for this functional test project, or click **Browse** to select a path.
4. Click **Finish**.

Connecting to a Functional Test project

If you have an existing Functional Test project or another member of your team has a project you need to use, you can use this option to connect to the project.

1. Go to the **Functional Test** perspective in IBM® Rational® Functional TesterHCL OneTest™ UI.
2. Click **File > Connect to a Functional Test project** from the menu bar.
3. In the **Project location path** field, type the path for the Functional Test project or click **Browse** to select the path of an existing Functional Test project.
   Result

   If you click **Browse** and select a path to an existing Functional Test project, Rational® Functional TesterHCL OneTest™ UI automatically enters the project name from the project location path.
4. In the **Project name** field, type a name to represent the Functional Test project (a logical name).

   The name must be unique.
5. Click **Finish**.
   A dialog box prompting you to upgrade the project from the older version and connect again is displayed.
6. Click **Yes** to confirm. Click **Cancel** if you want to disconnect from the project.

Disconnecting a Functional Test project

If you no longer need to work on a Functional Test project, you can remove it from the Projects view.

1. In the Projects view, right-click the project you want to disconnect.
2. Click **Disconnect**.

   Note: Disconnecting a project does not remove any files from the file system. You can reconnect the project at any time.

   Result

   The project is removed from the Projects view.

   To include the project in the Projects view again, connect on page 859 to the project.

Deleting a Functional Test project

There may be times that you want to delete a project that is no longer needed from the Functional Test Projects view.

1. Right click on the project which you want to delete in the **Functional Test Projects** view.
2. In the pop-up menu, click **Delete**.

Result

Rational® Functional TesterHCL OneTest™ UI displays a message asking you to confirm the deletion.

3. Select **Yes**.

Rational® Functional TesterHCL OneTest™ UI Projects view

The Functional Test Projects view, which is the left pane of the Functional Test Perspective, lists test assets for each project.

The following icons appear in the Projects view pane:

 Folders

 Simplified test scripts

 Java test scripts

 Shared test object maps

 Log folders

 Logs

 Java™ file

The Functional Test Projects view banner has the following buttons:

The **Connect to a Functional Test Project**  button allows you to connect to an existing Functional Test project.

The **Refresh Projects** button  enables you to repaint the display to reflect changes.

The **Synchronize with Editor** button  scrolls in the tree hierarchy to the name of the script currently displayed in the Java™ Editor.

Double-clicking a script in the Projects view opens the script in the Java™ Editor.



Note: If there are no projects in the Projects view, instructions display informing you how to create a new Functional Test project or connect to an existing Functional Test project. If you do not select any item in the Projects view, and right-click in the Projects view, a menu is displayed, from which you can create a new Functional Test project, connect to an existing Functional Test project, or refresh the project(s).

Right-clicking on a project or test asset listed in the Projects view displays various menu options, which are listed here in alphabetical order:

Add Empty Script -- Displays the Create an empty Functional Test script dialog box, which enables you to create a script on page 879 you can use to manually add Java™ code.

Add Script Using Recorder -- Displays the Record a Functional Test script dialog box, which enables you to enter information about the new script and start recording on page 877.

Add Test Folder -- Displays the Create a Test Folder dialog box, which enables you to create a new Functional Test folder on page 862 for the project or under an existing folder.

Add Test Object Map -- Displays the Create a Test Object Map dialog box, which enables you to add a new test object map to a project.

Add Test dataset -- Displays the Create a Test dataset dialog box, which enables you to create a new test dataset. on page 943

Clear As Project Default -- Removes the default designation from the selected test object map. To set the default designation, right-click the test object map in the Rational® Functional TesterHCL OneTest™ UI Projects view and select **Set As Project Default** .

Debug -- Launches the current script and displays the Test Debug Perspective, which provides information as the script debugs.

Delete -- Enables you to delete the selected test asset..

Disconnect Project -- Disconnects a Functional Test project on page 859, which removes it from the Functional Test Projects view.

Export -- Enables you to export project items for the selected log, project, or script.

Failed Verification Points -- Opens the selected verification point actual results file in the Verification Point Comparator on page 923, where you can compare and edit the data. See About Logs on page 1497.

Final Screen Snapshot -- Available when the log of a script that failed on its last run is selected. Opens the screen snapshot image taken at playback failure. See Screen snapshot on playback failure on page 1450.

Import -- Enables you to import project items for the selected log, project, or script.

Insert as "callScript" -- Available when a script is selected, inserts the `callScript ("scriptname")` code in the current script at the cursor location. See Calling a Script from a Functional Test Script on page 891.

Insert contained scripts as "callScript" -- Available when a project is selected, displays a message that enables you to choose **Yes** or **No** . **Yes** inserts a callScript command for all scripts in the project, including the selected folder(s) and all subfolders. **No** inserts a callScript command only for scripts in the selected folder(s). See Calling a Script from a Functional Test Script on page 891.

Merge Objects into -- Displays the Merge Test Objects into the Test Object Map page, which enables you to merge multiple test object maps.

Open -- Opens the selected script or Java™ class in the Java™ Editor or opens the selected test object map.

Open Log -- Opens the selected log. See About Logs on page 1497.

Open Test Object Map -- Enables you to display the selected test object map.

Properties -- Displays information about the selected Functional Test project, test object map, test folder, script, or log.

Rename -- Displays the Rename dialog box.

Reset Java Build Path -- Synchronizes the .JAR files in the Customization folder (C:\Program Files\IBM\Rational \FunctionalTester\Customization) (C:\ProgramData\HCL\UI\Customization) with the Java™ build path for Functional Test projects. The Java™ build path appears on the Java™ Build Path page of the Properties dialog box. For information, see the online *Java™ Development User Guide*.

Run -- Plays back a selected Functional Test script on page 1445.

Set as Project Default -- Indicates the selected test object map as the default in a variety of wizards and dialog boxes, such as the Select Script Assets on page 1834 page of the Record New Functional Test Script and the Create Empty Functional Test Script wizards, and the Copy Test Objects to New Test Object Map on page 1745 page of the Create new Test Object Map wizard. To remove the designation, right-click the test object map in the Projects view and select **Clear As Project Default** .

Show in Navigator -- Reveals the currently selected element's underlying resource (or the current editor's underlying resource) in the Navigator view. For information, see the online *Java™ Development User Guide*.

Team -- Enables you to add test elements to source control, check out elements, check in elements, undo a checkout, get latest version, show checkouts, display the history of an element, share a project, or compare versions or elements.

To open: Rational® Functional TesterHCL OneTest™ UI automatically displays the Projects view (by default) in the Functional Test Perspective.

Creating a new functional test folder

You can use folders to organize items in the Functional Test Projects view.

1. Display the Create a New Test Folder dialog box on page 1745 in any of these ways:
   - On the product toolbar, click the **Create a Test Folder** button  .
   - In the Projects view, right-click a project and click **Add Test Folder**.
2. In the **Enter or select the folder** field, either enter the appropriate path to the folder you want to create or use the navigation tools (**Home**  , **Back**  , and **Go Into** ) to select the path in the selected project.
3. In the **Test folder name** field, enter a folder name.
4. Click **Finish**.

Result

Rational® Functional TesterHCL OneTest™ UI adds the folder to the Functional Test Projects view in the path you selected.

Exporting functional test project items

You can export project items such as scripts, test object maps, Java™ files or Visual Basic files, datasets and application visuals to another functional test project.

1. Select the project in the Project view and click **File > Export** from the product menu. In the Export wizard, select **Functional Test > Functional Test Project Items** and click **Next**. Right-click the project in the Solution Explorer and click **Export**.
2. In the Select Items to Export page, verify that the items you want to export are selected and those you do not want to export are cleared.

   Notes:
   - The project items can be from any project. The project items are from the project you select.

3. To export the application visuals, select **Export Application Visuals**.
4. In the **Specify the export destination** field, enter the name or browse for the data transfer file, which is the file to export with the selected project items. If the file does not exist, Rational® Functional TesterHCL OneTest™ UI creates it with a .rftjdtr .rftvdtr extension.
5. Click **Finish**.

   Result

   Rational® Functional TesterHCL OneTest™ UI compresses a copy of the files into a data transfer file with the name you specified and appends a .rftjdtr .rftvdtr extension.

   When you export a script, Rational® Functional TesterHCL OneTest™ UI includes any necessary files, such as shared test object maps, even though you did not select them.

   To view items in the data transfer file, you can use any file compression program that supports the .zip format.

Importing functional test project items

You can import project items such as scripts, test object maps, Java™ files or Visual Basic files, and datasets into a Functional Test project.

1. Select the project in the Project view and click **File > Import** from the Functional Test menu. In the Import wizard, select **Functional Test Project Items** and click **Next**.Right-click the project in the Solution Explorer and click **Import**.
2. In the **Transfer file** field of the Import project items page, enter or navigate to the data transfer file name that was used to export the project items.

To view and work with items in the data transfer file, you can use any file compression program that support the .zip format. You do not have to extract files in the .rftjdtr .rftvdtrextension file before importing.

3. In the **Select the import location** field, use the navigation buttons to select the Functional Test project into which to import project items.
4. Click **Finish**.

   If the project already contains any of the assets you are importing, Rational® Functional TesterHCL OneTest™ UI displays the Select items to overwrite page.

5. Select the items that you want to overwrite in the project or clear the items that you do not want to overwrite and click **Finish**.
   - When you overwrite a script, Rational® Functional TesterHCL OneTest™ UI overwrites all the necessary files associated with the script, such as the test object map and the dataset. Other files in the project associated with the overwritten script, such as verification points, are deleted.

   Rational® Functional TesterHCL OneTest™ UI adds all the project items from the data transfer file into the project you specified.

Functional test assets

This section describes the different types of functional test assets that are available in Rational® Functional TesterHCL OneTest™ UI.

A typical Rational® Functional TesterHCL OneTest™ UI test script object includes these files:

- Script file (*scriptname*.java for Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, or *scriptname*.vb for Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration)

This file is created through recording.

- **Script helper file (*scriptname* ScriptHelper.java for Rational Functional TesterHCL OneTest UI, Java Scripting, or *scriptname* ScriptHelper.vb for Rational Functional TesterHCL OneTest UI, VB.NET Scripting)**

Each script has a script helper file that is generated after recording.

- Shared test object map file (*kadov_tag{{<ignored>}}filename*.rftmap kadov_tag{{</ignored>}}) or private test object map file (*scriptname*.rftxmap)

Each script has a map file. The map file can be associated with only one script (*. rftxmap) or shared among many scripts (*. rftmap). To prevent users from accidentally selecting a private map name as a shared map, the suffixes are different.

- **Verification Point file (*verificationpointname* .rftvp)**

Each script may also contain one or more verification point files. Verification point files are not shared among scripts.

- Script Definition file (*scriptname*.rftdef)

Each script contains a script definition file. The script definition file contains the name of the map file, script name, the names of all of the recognized objects, and other file linkage information.

- Public Test dataset (*filename*.rftdp) or Private Test dataset (*scriptname*.rftxdp)

You can associate a public or private test dataset with a test script. You can associate a public test dataset with one or several test scripts.

Working with functional test scripts (Windows-only)

This section describes the process of creating functional test scripts to test your applications.

Simplified scripting

Simplified test scripts are functional test scripts in the form of simple English statements that are easy to understand and edit. This feature is enabled by default in the Rational® Functional TesterHCL OneTest™ UI Preferences window.

When you record actions on the test application using the recorder, the functional test script is generated and displayed as a simplified test script in the Script editor. With the simplified test script feature enabled, you can also view the corresponding Java test script in the Java script editor. When you edit the simplified test script, the Java script reflects the changes in the Java script editor but not vice versa. You have the option to switch to the Java script editor using the **Insert Java Code Snippet** or **Insert Java Method** features available in the simplified test script editor and start working with the Java test script.

Note: You cannot migrate the existing functional test scripts that are generated as Java test scripts to simplified test scripts.

The Application View displays the captured application visuals (screen snapshots). You can click each test line of the simplified test script to view the application control it refers to. Rational® Functional TesterHCL OneTest™ UI captures the application controls and their properties during recording if the application visuals feature is enabled.

You can view the details and modify the properties of each test line such as an action on the application control or the test line arguments in the Properties View. For each test line of the simplified test script, you can also set the playback parameters such as execution delay and specify the log information such as types of messages in the Properties View.

Enabling simplified scripting

Simplified scripts are functional test scripts in the form of simple English statements that are easy to understand and edit. When you record actions on the test application using the recorder, the functional test script is generated and displayed as a simplified test script in the Script editor.

Before you begin

Note: This feature is enabled by default in the Rational® Functional TesterHCL OneTest™ UI Preferences window. You can override this setting for an individual script by selecting **Java Scripting** from the **Select Mode** list in the **Record a Functional Test Script** dialog box when you record the script.

1. Click **Window > Preferences**.
2. In the **Preferences** window, click **Functional Test > Simplified Scripting**.
3. Select **Enable Simplified Scripting**.

   Note: With the simplified scripting feature enabled, you can view the corresponding Java test script in the Java script editor and also work with the Java scripting.

4. Click **Apply**, and then click **OK**.

Creating a simplified test script

Use the recording feature of Rational® Functional TesterHCL OneTest™ UI, to record actions on the test applications. The actions on the test applications are generated in the form of a simplified test script when you stop recording. The generated simplified test script is displayed in the simplified **Script** editor in Rational® Functional TesterHCL OneTest™ UI. You can also view the corresponding Java™ code of the recorded test script in the Java™ script editor.

Before you begin

Prerequisites:

- The **Enable Simplified Scripting** option is enabled on the Functional Test Preferences page.
- The test application and the required environments must be configured for functional testing.
- A functional test project is created.

1. Click **File > New > Functional Test Script Using Recorder**.

   The **Record a Functional Test Script** page is displayed.

2. Select the project to associate the script and then enter a name for the script.
3. **Optional:** Select **Add the script to Source Control** option to add the script to a source control.
4. Select the **Simplified Scripting** option in the **Select Mode** list, if it is not already selected.

   Note: You can make simplified scripting as the default script mode by selecting the **Default** checkbox.

5. Click **Next**.

   The **Select Script Assets** page is displayed.

6. Modify the values of the test object map, helper superclass, or test dataset fields, if you do not want to use the default settings.
7. Click **Finish** to start recording.

The **Recording Monitor** is displayed and the recording starts.

8. Click **Start Application** icon in the **Recording Monitor** toolbar to start the test application.

   Notes:

   - If the HTML test application has UI controls and the controls open a new window, you must perform any of the following actions to enable Rational® Functional TesterHCL OneTest™ UI to capture the user actions on UI controls in the new window:
     - Click any area within the window.
     - Click the **Maximize** button on the window.
   - If you want to record verification points or data-drive your test script, you can use the tools available in the **Recording Monitor** toolbar.
   - If you want to insert statements to call another script, specify log information, timer or comments during recording, you can click **Insert Script Support Commands** in the **Recording Monitor** toolbar.

9. Perform the actions in the test application that you want to record.

   The recording monitor displays the actions that you performed.

10. Click **Stop recording** ( ) when you finish recording.
    Result
    A simplified test script is generated and displayed in the simplified **Script** editor.

Editing a simplified script

You can edit a test line in the script editor and also modify the test line properties, such as the control actions in the Properties View.

About this task

In the script editor you can edit a test line that contains input values.

1. Select a test line in the script editor.
2. Modify the input values.

3. Click **File > Save** to save the modified script.



Tip: You can also modify the input values on the **General** page in the **Properties** view.



Note: You can cut, copy, and paste the test lines in the script editor. These options are available in the **Edit** menu.

Modifying the control name and the action

The **General** page in the **Properties** view displays the details of the test line that is selected in the script editor. On the General page, you can modify the test-line details such as the control name and the action.

1. Select the test line in the script editor.
   Result
   The test line details are displayed in the **Properties** view.
2. Click the **General** tab in the **Properties** view.
   The **General** page opens.
3. Modify the **Control name** value.
   Result
   The test line in the script editor also reflects the modified value.
4. To modify the action on the control, select the required action from the list. The **Action** list displays all the actions that you can perform on the selected control.
   Result
   The test line statement in the script editor is modified.
5. To modify the input values or the action details such as the screen coordinates or the path, modify the values in the **Action Parameters** field.
6. Click **File > Save** to save the modified script.

Disabling a test line

You can disable a test line in the script editor so that the test line is not run during script playback.

1. Select the test line in the script editor.
2. Right-click, and select **Enable/Disable action.**
   The test line font changes to italics, indicating that the test line is disabled.
3. Click **File > Save** to save the modified script.

Grouping test lines

The test lines in the script editor are grouped based on the parent window that the test line control refers to. You can create more logical groups to manage the test lines for easy identification. For example, if you have multiple tabbed pages in a window, by default all the test lines are grouped into a main window group. You can create multiple groups within the main group for listing all the actions on the controls based on the tabbed pages.

1. In the script editor, select the test lines to group together.
   Press and hold the Ctrl key while selecting multiple test lines.
2. Right-click, and select **Create Group** ( ).
   A group title is added and all the selected test lines are grouped together. By default, the main parent window name is added as the group name. You can change the group name, if required.
3. Click **File > Save** to save the modified script.

Inserting comments in the script

You can add comments in the simplified script editor.

1. Select the test line in the script editor.
2. Right-click, and select **Insert Comment** ( ).
   A test line with this text is inserted into the script: `//Comment`.
3. Type your comments in the comment line.
   The comment text is displayed in italics. The comment lines are not executed during playback.

   Note: You can only specify single-line comments. Multiple-line comments are not supported.

4. Click **File > Save** to save the modified script.

Repeating actions

You can repeat the action statements in the simplified script.

1. Select the test line or a group in the script editor.
   Press and hold the Ctrl key while selecting multiple test lines.
2. Right-click, and select **Repeat action** ( ).
   A `Repeat` title is added and the selected test lines are grouped into the Repeat group.
3. In the **Properties-General** view, specify the number of times to run the statements in the **Repeat Count** field.
4. Click **File > Save** to save the modified script.
   Result
   During script playback, the test lines in the Repeat group are run based on the repeat count.

Inserting conditional statements

You can insert conditional statements to verify the values of the variables in the script and perform actions in the application.

Before you begin

All variables to be tested while running conditional statements must be captured during the script recording. To do this, you must use the Get a Specific Property Value feature that is available in the verification point wizard while recording the script to assign a control property to a variable in the script.

1. Select the test line in the script editor.
2. Right-click and select **Insert Condition (If Clause)** ( ).
   Result
   An **If Then** clause is inserted in the script editor. The selected test line is inserted in the Then group.
3. Select the **If** clause.
4. In the **Properties-General** view, specify the variable values that must be verified.
   a. Select the variable that must be verified in the **Left Side** field.
      The **Left Side** field lists all the variables that are declared during the script recording before the selected test line.
   b. Select the required parameter in the **Compares To** field.
   c. Type the variable value in the **Right Side** field.
      For string values, you must specify the value using quotation marks, for example, `"Visa"`. You can also select another variable from the list if the first operand must be verified against another variable.
5. **Optional:** To run other test lines that must also run if the variable conditions are met, select and drag the test lines into the **Then** group.
6. You can insert the Else clause for the test lines that must be run if the variable conditions are not met. Select the `If` or `Then` clause, or any test lines in the `Then` group, right-click and select **Insert Else Clause** ( ).
   Select and drag the test lines that must be executed if the variable conditions are not met into the **Else** group.
7. Click **File > Save** to save the script.
   Result
   During script playback, the variable conditions are checked and the required test lines are run.

Exemple

In this example, a conditional statement `if (Item_text EQUALS "Schubert")` has been inserted into the functional test script. The test lines grouped under the `Then` group are run only when the value of the variable Item_text is `Schubert`. This script, when run on the ClassicsJavaA application, will place the order only if the value of the variable Item_text is `Schubert`.

```
Start  Application  ClassicsJavaA
ClassicsCD
  Click  tree2  at  Composers->Schubert->Location(PLUS_MINUS)
  Click  tree2  at  Composers->Schubert->String Quartets Nos. 4 & 14
  Click  Place Order
Member Logon
  Click  OK
Place an Order
  Get  Property  Item:   text
  if (Item_text EQUALS "Schubert")
    Then
      Click  Card Number (include the spaces)
      Type  Value  12345678
      Click  Expiration Date
      Type  Value  12/12
      Click  Place Order
```

```
        Message
          Click  OK
  ClassicsCD
    Close  ClassicsCD
```

Specifying the playback options for a simplified script

You can set the wait time for a control to be displayed or pause an execution of a test line for a simplified script.

Before you begin

The test application might take some time to load or refresh the controls during script playback. This might result in an exception or playback failure. You can set the options to wait for a control or pause an execution of a test line.

To wait for the control to be displayed in the test application during playback:

1. Select the test line in the script editor.
   Result
   The **Properties** view displays the details of the selected test line.
2. Click the **Playback** tab in the **Properties** view. Type the time in seconds in **Wait for the control to be displayed**.
3. Click **File > Save** to save the changes.
   Result
   During script playback, the action on the control that the test line refers to is performed only after waiting for the control to be displayed in the test application for the specified time.

Pausing an execution of a test line

About this task

You can pause or delay running a test line during script playback. You can set this option for delaying the run of the test line so that the application is started or wait for the application to populate the data in the control.

1. Select the test line in the script editor.
   Result
   The **Properties** view displays the details of the selected test line.
2. Click the **Playback** tab in the **Properties** view. Type the time in seconds in the **Pause execution for** field.
3. Click **File > Save** to save the changes.
   Result
   During script playback, running the test line is delayed for the specified time.

Handling exceptions during script playback

If an exception occurs during script playback, you can specify how Rational® Functional TesterHCL OneTest™ UI handles these exceptions.

Before you begin

Various situations can cause playback exceptions. Some common exceptions are as follows: a control is not found, a control is ambiguous, and a weak recognition. You can set the conditions for handling exceptions during script playback for each test line in the script.

To set the conditions for handling the exceptions for a test line during script playback:

1. Select the test line in the script editor.
   Result
   The **Properties** view displays the details of the selected test line.
2. Specify the type of action that must be performed if an exception occurs while a test line is running during script playback.
   You can select any action such as **Stop**, **Skip and Continue** , or **Retry** from the list for any of the following exceptions:
   - **Control not found**
   - **Ambiguous control**
   - **Weak recognition**
   - **Control sub item not found**
   - **Unexpected error**
3. Click **File > Save** to save the changes.

Specifying the log details for a test script

You can specify the message to be displayed in the log for a test line that has run. You can also specify the type of screen capture that must be taken during playback and displayed in the log so that you can view the state of the control or the screen in the test application.

1. Select the test line in the script editor.
   Result
   The **Properties** view displays the details of the selected test line.
2. Click the **Log** tab in the **Properties** view.
3. To capture a snapshot of the control or the screen in the test application while running the test line, select either **Control Snapshot** or the **Screen Snapshot** option.
4. To display a message in the log for a test line that has run, indicate whether to display an information, warning or an error message with the description.
5. Click **File > Save** to save the changes.

Deleting a test line

You can delete a test line from a script in the script editor. The application visual that is captured and displayed while recording the test script is not deleted from the Application view.

1. Select the test line in the script editor.
2. Right-click and select **Delete action (**  **)**.
3. Click **OK** in the confirmation message window.

Result

The test line is deleted from the script.

Note: You can undo the delete actions to restore the test lines in the script. You also have the option to redo the delete actions.

Working with application visuals

Application visuals are snapshots of the controls or screens of the test application that are captured while recording scripts. The visuals of the test application are captured only if both the simplified scripting and the application visuals feature preferences are enabled.

Application visuals provide a quick way to view application controls and also modify the script without opening the test application.

Using the application visuals, you can modify the test script to test additional application controls, create or edit verification points, or insert data-driven commands into the script without opening the application under test.

Note: The application visuals feature works only with JRE version 1.4 and above.

Enabling application visuals

You can enable the application visuals feature so that Rational® Functional TesterHCL OneTest™ UI captures the application controls and their properties during recording. You can also specify whether the verification points or data-driven command feature in the application visuals must be enabled so that you can insert those elements into the script using the application visuals without opening the test application.

Before you begin

Prerequisite: The simplified scripting feature must be enabled. The application visuals feature is not available for Java scripting.

1. Click **Window > Preferences**.
2. In the **Preferences** window, click **Functional Test > Simplified Scripting > Application Visuals**.
3. Select **Enable Application Visuals**.
   By default, this feature is enabled with the simplified scripting.
4. **Optional:** You can enable the following application visuals option so that you can insert verification points and data driven commands into the script using application visuals:
   a. Select **Insert Data Driven Commands** to insert the data-driven commands into the script from application visuals.
   b. Select **Show Verification Point Dialog** to use the verification point wizard while inserting the data verification points into the script using application visuals.
   c. Select **Enable capturing of verification on test data** to insert verification points into the test script using application visuals.
5. Click **Apply**, and then click **OK**.

Inserting an application control into the script by using an application visual

You can modify a test script to test additional application controls by inserting controls into the script by using an application visual.

Before you begin

Prerequisite: The visuals of a test application are captured only if both the simplified scripting and the application visuals feature preferences are enabled during recording. The **Application View** displays the application visuals that can be used for inserting additional controls into the script for testing without opening the test application.

About this task

To insert a control into a test script:

1. Select the test line in the script editor that refers to the required application visual of the test application. The application visual that contains the control is displayed in the **Application view**.

   Tip: The thumbnail pane in the Application view displays images of all the application visual that are captured for the project. You can select the required application visual image in the thumbnail pane and view the application visual in the Application view.

2. In the Application view, point to the control that you want to insert into the test script. The control is highlighted in red.
3. Right-click and select **Insert *control name* control > *action on the control***.
   The actions that can be performed on the control are listed when you select a control in the Application view.
   Result
   The test line for performing the action on the control is inserted into the test script. You can drag the test line in the script editor to get the required sequence of test lines in the script.

   Tip: To insert a control into the script, you can also drag the control from the application visuals to the script editor. The test line for performing the default action on the control is inserted into the test script.

4. Click **File > Save** to save the modified test script.

   Note: You can change the action on the control or set the playback or log details for the inserted test line in the Properties view.

Updating the application visuals in the Application view

The application visuals are captured when you record simplified scripts. If the test application is changed after you record the script, you can update the current application visuals with updated visuals from the test application.

Before you begin

Prerequisite: The required window or the user interface of the test application must be opened.

1. Select the test line in the script editor that refers to the required application visual of the test application. The application visual is displayed in the Application view.
2. Right-click the application visual, and click **Update Visual**.
3. On the Select an Object wizard page, click the **Object Finder** ( ) and drag it into the application over the control to add to the Application view. For other methods of selecting objects, see Select an Object on page 1823.
4. On the **Update Visual Objects** page, all the application visuals that are captured from the selected window or user interface are displayed. Select the visual object to update from the list.
5. Click **Finish**.

   Result

   The application visual in the Application view is updated with the application visual from the test application.

Switching to Java scripting

You can switch to Java scripting to insert Java code to perform additional operations such as extending an API or functions that cannot be performed directly in the simplified script editor. To use both the simplified script and Java scripting, you must use the Insert Java Code Snippet or Insert Java Method feature that is available in the simplified script editor and switch to Java scripting. If you modify the Java script directly without using these features, the Java script changes are lost and the simplified script runs during playback.

Use the Insert Java Method feature to insert a method to logically group the lines of code. You can call the Java method from different locations in the script using the Java Code Snippet feature.

Use the Insert Java Code Snippet feature to insert a few lines of code.

When you insert Java custom code, a test line in the simplified script editor indicates that Java code or a Java module is inserted in the Java script editor. The Java code snippets are added between the test lines as indicated in the simplified script editor. A block of comments is added in the Java script editor, indicating the start and the end points for adding the Java code. The Java method is added at the end of the Java script. During playback, the Java code snippets and the Java method are run.

Note: If you have generated simplified scripts and switch to Java scripting mode permanently, you can edit the Java code outside the start and the end points and continue working with Java scripting. Ensure that you do not edit the simplified scripts. All the changes in the Java code are lost and the updated simplified script is run during playback.

Inserting a Java code snippet

You can insert a Java code snippet to perform additional operations that are not supported by the simplified scripts. The Java code snippet runs during script playback.

Before you begin

Prerequisite: Knowledge of Java programming

1. Select the test line in the simplified script editor. Right-click and select **Insert Java Code Snippet ( )**.
   Result
   This line is inserted after the selected test line: `Click here to tag the Java snippet`
2. To easily identify the Java code in the Java editor, select the inserted test line and replace the test line text by typing a tag for the Java code.
3. Click **File > Save** to save the simplified script.
4. Click **Java** editor that is displayed next to the **Script** editor.
   Result
   The inserted Java tag is displayed as a comment with the start and the end point for inserting the Java code in the Java editor.
5. Type the Java code within the start and the end comment section.
6. Click **File > Save** to save the Java script.
   Result
   During the script playback, the inserted Java code also runs.

Note: Do not edit the Java code outside the start and the end points that are inserted when you use the Insert the Java code snippet feature in the simplified script editor. All the changes to the Java script are lost if you later edit the simplified script.

Inserting a Java method

You can insert a Java method to perform additional operations that are not supported by the simplified scripts. The Java method is added at the end of the Java script. You must insert a Java code snippet and call the Java method.

Before you begin

Prerequisite: Knowledge of Java programming

1. Select the test line in the simplified script editor. Right-click and select **Insert Java Method ( )**.
   Result
   The test line `JavaModule` is inserted after the selected test line.
2. Click **File > Save** to save the simplified script.
3. Click **Java** editor that is displayed next to the **Script** editor.
   Result
   The Java method is displayed as a comment with the start and end points for inserting the Java code in the Java editor.
4. Type the Java code within the comment section.
5. Click **File > Save** to save the Java script.
   Result
   During script playback, the inserted Java code runs.

Notes:

- You must call the Java method from a Java code snippet. The Java method runs when the Java code snippet runs.
- You can also call the Java method from a different script within the functional test project.
- Do not edit the Java code outside the start and end points that are inserted when you use the Insert Java Method feature in the simplified script editor. All the changes to the Java script are lost if you later edit the simplified script.

Exporting a simplified script

You can export the simplified scripts in a .txt, .xml or .html format so that the script can be viewed outside Rational® Functional TesterHCL OneTest™ UI.

1. Click **File > Export**.
2. On the **Export** wizard page, click **Functional Test > Simplified Scripts**, and then click **Next**.
3. Select the simplified scripts to export.
4. Select the file type for the simplified script in the **Output format** field.
5. Specify the location to export the simplified script.
6. Click **Finish**.

Java scripting

When you record a functional test script to test an application, simplified scripts are generated and displayed in the Script editor. The corresponding Java test script is displayed in the Java editor.

While working with the simplified scripts, you can switch to Java scripting to use some of the Java functions such as APIs by using the Insert Java snippet and Insert Java module features available in the simplified script editor. Advanced users can work with Java scripting instead of working with the simplified test scripts.

The simplified scripting feature is enabled by default in the Rational® Functional TesterHCL OneTest™ UI preferences page. To work with Java scripting, you can disable the simplified scripting feature. The application visuals feature is not available while working with Java scripts, you must instead use the functional test object maps.

Recording a Java™ test script

Java™ scripts are generated when you work with the simplified scripts. You can switch to Java™ scripting using the Insert Java™ snippet and Insert Java™ module features in the simplified script editor.

Before you begin

Advanced users can opt to work solely with Java™ scripting. Use the recording feature of Rational® Functional TesterHCL OneTest™ UI to record actions on the test applications. The actions on the test applications are generated in the form of a Java™ test script when you stop recording.

Prerequisites:

- Disable the Simplified Scripting feature on the Rational® Functional TesterHCL OneTest™ UI Preferences page. Alternatively, if you want to work with Java™ scripting only for the current script you will record, select **Java Scripting** from the **Select Mode** list in the **Record a Functional Test Script** dialog box.
- The test application and the required environments must be configured for functional testing.

  Rational® Functional TesterHCL OneTest™ UI automatically enables the environments for functional testing. As a result, you can directly record functional test scripts without enabling components manually. The automatic enablement takes place under certain conditions and has limitations. For more information about the conditions and limitations, see Automatically enabled environment for functional testing on page 774.
- A functional test project has been created.

About this task

All functional testing scripts use a default helper superclass. You can create your own helper superclass if you want to add additional methods or override the methods in RationalTestScript. For more information, see Changing the Default Script Helper Superclass on page 887.

1. Click **Record a Functional Test Script**( ).
2. In the Record a Functional Test Script dialog box on page 1800, select the project for the script to be a part of, and then type a name for the script.

   Note:
   - Script names cannot contain spaces or the following characters: `$ \ / : & * ? " <> | # % -`
   - The script name is appended to the project path. The project path and script name together should not exceed 230 characters.

3. Make sure that **Java Scripting** is selected in the **Select Mode** list. You can make Java™ scripting as the default script mode by selecting the **Default** check box.
4. Click **Next**.
5. **Optional:** In the **Select Script Assets** page, modify the test object map and test dataset if you do not want to use the default settings.
6. Click **Finish** to start recording.
   The **Recording Monitor** opens and the recording starts.
7. In the **Recording Monitor** toolbar, click to start your test application.
8. Perform test actions in the application.

   Note: To record the action of moving the mouse over a link that has a tooltip, move the mouse over the link so that the tooltip is displayed, and press Shift. This notifies the recorder to capture the action in the script.

9. **Optional:** You can record verification points or data-drive your test script using the tools available in the **Recording Monitor** toolbar while recording the script.

10. **Optional:** To insert statements to call another script, specify log information, timer or comments during recording, use the **Insert Script Support Commands** feature available in the **Recording Monitor** toolbar.
11. Close your application, if you want closing the application to be part of the script.
12. Click **Stop recording** ( ) when you finish recording.
    Result
    A Java™ test script is generated and displayed in the **Java** editor.

Creating a new test script without recording

As an alternative to recording, you can create a script to enter Java™ code manually.

About this task

In the new script, Rational® Functional TesterHCL OneTest™ UI includes import statements for files you need to compile the script and comments that contain archiving information. Rational® Functional TesterHCL OneTest™ UI uses the script name as the class name and set up the testMain file, where you can add the commands to include in the script.

All functional test scripts use a default helper superclass. You can create your own helper superclass if you want to add additional methods or override the methods in RationalTestScript. For more information, see Changing the Default Script Helper Superclass on page 887.

1. Click **Create an Empty Functional Test Script** ( ) on the Functional Test toolbar.
2. In the Create an empty Functional Test script dialog box on page 1747, enter or select a folder for the script and type a name in **Script name**. The script name must be a valid Java™ class name.
3. Check **Add the script to Source Control** if you want the script to be under source control. For more information, see About Software Configuration Management on page 305.
4. Take one of the following steps:

    a. (Optional) To use a different test object map, helper superclass, or test dataset, click **Next**.

    In the Select Script Assets page on page 1834, select any of the following test assets and click **Finish**:
    - Select and set a default test object map.
    - Select and set a default helper superclass.
    - Select a test dataset.
    - Select a dataset record selection order.

    b. To create the new script, click **Finish**.
    Result

    Functional Test creates a local test object map for your script and displays the script in the Projects View.

5. Start adding code to the script.

    You can use the Test Object Map to add objects and methods to the script.

To insert any features into the script, such as a call script command, log entry, timer, script delay, or comment, click **Insert Recording into Active Functional Test Script** () on the Functional Test toolbar. On the Recording toolbar, click **Insert Script Support Commands**().

You can also use buttons on the Recording toolbar to start an application from the script or create a verification point.

Creating a test script without recording in the Visual Studio IDE

As an alternative to recording, you can create a script to enter VB.NET code manually.

About this task

Rational® Functional TesterHCL OneTest™ UI includes import statements for files you need to compile and comments that contain archiving information. Rational® Functional TesterHCL OneTest™ UI uses the script name as the class name and sets up TestMain, where you can add the commands to include in the script.

All Functional Test scripts use a default helper base class. You can create your own helper base class if you want to add additional methods or override the methods within RationalTestScript. For more information, see Changing the Default Script Helper Base Class.

To create a new Functional Test script manually:

1. Click the **Create an Empty Functional Test Script** button  on the Functional Test toolbar or click **File > New > Add Empty Script** or **Project > Add Empty Script**.
2. In the Add New Item dialog box, type a script name in the **Name** field or click **Browse** to select a folder for the script in the **Location** field. The name must be a valid Vclass name.
3. Click **Open** to display the Select Script Assets page on page 1834.
   Result

   Rational® Functional TesterHCL OneTest™ UI creates a local object map for your script.
4. (Optional) To use a different test object map on page 1838, helper base class on page 1829, or test dataset on page 938, select them from the Select Script Assets page. You can also set the dataset record selection order and set a default test object map and helper base class.
5. Click **Finish** to create the new script.
   Result

   Rational® Functional TesterHCL OneTest™ UI displays the script in the Solution Explorer.
6. Start adding code to the script.

   You can use the Test Object Map to add objects and methods to the script.

To insert any features into the script, such as a timer, call script command, log entry, script delay, or comment, click the **Insert Recording into Active Functional Test Script** button on the product toolbar. On Recording toolbar, click the **Insert Script Support Commands** button .

You can also use buttons on the Recording toolbar to start an application from the script or create a verification point. Click the **Display Help** button on the Recording toolbar for information about the options on the Recording toolbar.

7. To put the script under source control:
   a. Right-click the new script in the Solution Explorer, and click **Check In**.
   b. In the Check In dialog box, click **Check In**. For more information, see About Software Configuration Management on page 305.

Recording scripts in an existing script

You can record new scripts in an existing script to start an application under test (AUT), insert verification points, and add script support functions.

Before you begin

You must have completed the following tasks:

- Recorded a test script. See Recording a script on page 902.
- Read and been familiar with creating verification points. See Verification points on page 904.
- Read and been familiar with using script services. See Script support functions on page 890.

1. Open the test script and place the cursor in the script where you want to begin a recording.
2. Click **Script > Insert Recording**.
   The **Recording** toolbar is displayed, and then the recording begins.
3. Click **Start Application** on the **Recording** toolbar.

   The **Start Application** dialog is displayed.
4. Select the AUT, and then click **OK**.
5. Perform any action in the application so that the actions are recorded.
6. **Optional:** Click **Insert Verification Point or Action Command** on the **Recording** toolbar to insert verification points on the required objects in the AUT.
7. Close the application.

   Note: You can close the application to record the close action.

8. **Optional:** Click **Insert Script Support Commands** on the **Recording** toolbar to insert any features such as a call script command, log entry, timer, script delay command, or comment into the script.

9. Click **Stop Recording** on the **Recording** toolbar to stop the recording.
   The Rational® Functional TesterHCL OneTest™ UI window is restored, and then the updated script is displayed.

Results

You have recorded a new test script in an existing script.

Recording scripts for an AUT with different user credentials

When an application under test (AUT) is configured with a different user credential by using a batch file on the Windows operating system, you can record scripts for the AUT.

Before you begin

You must have completed the following tasks:

- Created a batch file that includes commands to start the AUT as a different user.
- Configured the batch file as an application for testing. See Configuring applications for testing on page 796.
- Read about creating a new test script without recording and recording in an existing script. See Creating a new test script without recording on page 879 and Recording scripts in an existing script on page 881.

About this task

You can either create a new test script or insert a recording in an existing script for an AUT that is configured with a different user.

To record the test scripts, you must ensure that you start Rational® Functional TesterHCL OneTest™ UI as an administrator. With the administrator privilege, you can record scripts for the AUT with different user credentials.

1. Start Rational® Functional TesterHCL OneTest™ UI as an administrator.
2. Perform any of the following actions:
   - Go to Step 3 on page 882 to create a new script.
   - Go to Step 4 on page 883 to insert the recording into an existing script, if you have an existing script.
3. Perform the following steps in the **Functional Test** perspective:

   a. Click **File > New > Functional Test Script Using Recorder**.
      The **Record a Functional Test Script** page is displayed.

   b. Select the project to which the script must be associated.

   c. Type a name for the script.

   d. Click **Finish**.
      The **Recording Monitor** toolbar is displayed.

   e. Go to step 5 on page 883.

4. Perform the following steps in the **Functional Test** perspective:
    a. Open the script.
    b. Place the cursor in the script where you want to insert the recording.
    c. Click **Script > Insert Recording**.
    The **Recording Monitor** toolbar is displayed.
    d. Go to step 5 on page 883.
5. Click **Start Application** on the **Recording Monitor** toolbar.
    Result
    The **Start Application** dialog box is displayed.
6. Select the batch file that is configured to start the AUT, and then click **OK**.
    Result
    The AUT is started for recording.
7. Perform the actions that you want to record.
8. Close the AUT.

    Note: You can close the AUT, if you want to record the close action.

9. Click **Stop Recording** when the recording is complete.

Results

You have recorded scripts for an AUT with different user credentials by using a batch file.

What to do next

You can play back the recorded test. See Running scripts from Rational Functional TesterHCL OneTest UI on page 1445.

Recording scripts to test HTML applications

Record scripts to test HTML applications on a single browser as you record any functional test script.

Before you begin

Important: If you enabled Mozilla Firefox or Google Chrome browser for IBM® Rational® Functional TesterHCL OneTest™ UI, the latest Java update must be associated with the browser. If not done, security messages prompt up when you open the browser and Java will be blocked.

About this task

There are two basic steps for recording scripts:

- Enable a web browser on page 780. Before you can use a web browser to test an application, you must enable it. Click **Configure > Enable Environments for Testing** to enable your environments for testing. You must enable the supported versions of Firefox, Microsoft Edge, or Internet Explorer browsers before you record functional tests for HTML applications.
- Configure your HTML applications for testing on page 796. Click **Configure > Configure Applications for Testing** to specify information about your application and its environment. If you plan to test Microsoft® HTML Applications (MSHTA), run mshta.exe to configure each application that you want to test.



Note: There are special considerations when recording cross-platform/cross-browser scripts on page 884.

Related information

Enabling Microsoft Edge to test HTML applications on page 782

Recording cross-browser and cross-platform scripts

This topic provides an overview of the procedures to set up your environment to record cross-browser scripts.

1. Click **Configure > Enable Environments for Testing** to enable browsers for testing. For example, any of the supported versions of Firefox or Internet Explorer that you plan to use for testing must be enabled.
2. Designate any one browser as a default:
    a. In the **Web Browsers** field, click the name of the browser.
    b. Click **Set as Default**.
    c. Click **OK**.
3. Click **Configure > Configure Applications for Testing** to specify information about your application and its environment.
4. Start recording a functional test script.
5. Start your test application from the Record toolbar. Rational® Functional TesterHCL OneTest™ UI opens the HTML page you specify in the default browser.
6. Perform any actions and create verification points on your test application. Stop recording.
7. Run the script.
   Rational® Functional TesterHCL OneTest™ UI plays back the script in the default browser.
8. Designate the other browser as a default:
    a. In the **Web Browsers** field, click the name of the other browser.
    b. Click **Set as Default**.
    c. Click **OK**.
9. Run the script. Rational® Functional TesterHCL OneTest™ UI plays back the script in the browser you defined as the default in step 8.

Displaying test object information

You use the Test Object Inspector to examine graphical components visible in the running application and display information about those objects, such as parent hierarchy, inheritance hierarchy, test object properties, non value properties, and method information.

1. Start the application that contains the objects you want information about.
2. Start the Test Object Inspector in either of two ways:
   - From the product menu, click **Run > Test Object Inspector Tools > Test Object Inspector**.
   - From the product toolbar, click the **Open Test Object Inspector** button  .

   The Test Object Inspector does not display information for the test object under the cursor.

   For an enabled Java™ or already-infested application, Test Object Inspector automatically tracks the cursor and performs live updates immediately after you open the application.

   Note: If the application is not active, Test Object Inspector does not capture objects in the application. You must pause on the application to force the infestation before Test Object Inspector can track the cursor and perform live updates against that application.

3. If the application is not Java-enabled or already-infested, hover the mouse over the object in the application, and press **Shift**.

   Test Object Inspector captures the object and copies the test object information displayed in the Test Object Inspector window to the system Clipboard.

   To select another test object, click the **Resume**  button and move your cursor to the other test object.
4. Use the **View** menu to select the type of information to display.

Getting a property value

You can get a single property value for the selected object while you are recording. It puts a getProperty into your script and returns the value during playback.

Before you begin

Prerequisites: The test application is started

About this task

This information is useful if you need to make a decision based on the property. For example, you might want to query whether a button is enabled.

1. Click the **Record a Functional Test Script** button on the product toolbar.
2. In the Recording Monitor, click the Start Application button to start your test application.
3. Locate the object in your application that you want to get a property for.

4. In the Recording Monitor, click the **Insert Verification Point or Action Command** button.
5. On the **Select an Object** page of the Verification Point and Action Wizard, use the Object Finder to select the object in your application. Once you have selected the object, click **Next**.
6. On the **Select an Action** page, click the **Get a Specific Property Value** option and click **Next**.
7. When you selected the object, the property list was automatically created and displayed in the **Property Name** and **Value** fields on the Insert getProperty Command Page. Select the property that you want to get. Click **Next**.
8. On the Variable Name page, verify the information listed in the **Object**, **Property**, and **Data Type** fields.
    a. In the **Variable Name** field, accept the default suggestion listed in this box, or type a new name. The default name is based on the name of the object and the property you are testing.
    b. The **Declare the variable in the script** option is selected by default. You need to declare a variable the first time you use the variable name. If you use the same variable name again in the same script, clear this option after the initial instance.
9. Click **Finish**.
   Result

   The statement containing the getProperty will then be written into your script at the point you inserted it.

Exemple

Example

If you get the **label** property on a button called Place Order, this is what would be written into your script:

```
String PlaceOrder_label = (String)placeOrder().getProperty("label");
```

If you get the **enabled** property on a button called Place Order, this is what would be written into your script:

```
Dim PlaceOrder_enabled As Boolean = PlaceOrder.GetProperty("enabled")
```

Setting a wait state for an object

This feature is used to set a wait state for an object during playback to check for its existence. This is useful when waiting for an object right after starting your application, or after other actions that may take a long time.

1. Click the **Record a Functional Test Script** button on the product toolbar.
2. In the Recording Monitor, click the **Start Application** button to start your test application.
3. Locate the object in your application that you want to wait for.
4. In the Recording Monitor, click the **Insert Verification Point or Action Command** button.
5. On the **Select an Object** page of the Verification Point and Action Wizard, use the Object Finder to select the object in your application. Once you have selected the object, click **Next**.
6. On the **Select an Action** page, click the **Wait for Selected Test Object** option and click **Next**.
    a. To set a wait state for the object, either use the defaults, or set your own time. **Maximum Wait Time** is the maximum number of secondsRational® Functional TesterHCL OneTest™ UI will wait for the object to appear in your application during playback. **Check Interval** is the number of seconds between times thatRational® Functional TesterHCL OneTest™ UI will check for the object during the wait period.

b. Check **Use the defaults**.Rational® Functional TesterHCL OneTest™ UI check for the existence of the object in your application every 2 seconds, for up to 120 seconds.
c. To set your own time, clear the **Use the defaults** check box and type in your own values for **Maximum Wait Time** and **Check Interval**.

7. Click **Finish**.
   The statement containing the waitForExistence will then be written into your script at the point you inserted the object.

Recording and playing back double byte characters on Chinese systems

The following information pertains to record and playback of DBCS on Chinese systems.

Simplified Chinese:

The only Input Method Editor (IME) that is supported for recording and playing back scripts using Rational® Functional TesterHCL OneTest™ UI on Simplified Chinese systems is the MS-PinYin IME. During script playback this IME will automatically be activated for the input of Chinese characters, provided the IME is present on your system. Use of other IMEs for recording is not supported and may yield unexpected results.

Traditional Chinese:

Two IMEs are supported for the recording and playback of Traditional Chinese characters: the New Phonetic and New ChangJie IMEs. During script playback the New Phonetic IME will automatically be activated for input of Chinese characters and if that is not present, the New ChangJie IME will be used. Use of other IMEs for recording is not supported and may yield unexpected results.

Changing the default script helper superclass

By default, all Functional Test scripts extend the RationalTestScript class, and thereby inherit a number of methods (such as callScript). Advanced users may want to create their own helper superclass which extends RationalTestScript and can add additional methods or override the methods from RationalTestScript.

About this task

You can specify a helper superclass that Rational® Functional TesterHCL OneTest™ UI will use whenever you create or record a script in your project. This default superclass is specified in the Functional Test Project Properties page. You can also specify a helper superclass for an individual script in the Functional Test Script Properties Page. Once a script has been created, it retains the reference to the default superclass as its own helper superclass.

To change the default script helper superclass for a project:

1. Select a project in the Projects view.
2. Right-click and select **Properties**.
3. Click **Functional Test Project**.

The Functional Test Project Properties page opens. The helper superclass listed here will be used by default for all new scripts created or recorded in this project.

Note: This is a user-specific preference and will not be shared by other users of this project.

4. To change the default superclass for the selected project, enter the fully-qualified class name of your custom helper superclass in the Default Script Helper Superclass field. Note that your helper superclass must extend RationalTestScript.

Results

Note: If you change your superclass and then want to reset it back to RationalTestScript later on, you can either type RationalTestScript in the superclass field or just clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

To change the script helper superclass for an individual script:

1. Select a script in the Projects view.
2. Right-click and select **Properties**.
3. Click **Functional Test Script**.

   The Functional Test Script Properties Page opens. The helper superclass listed here will be used for the script you selected.

4. To change the default superclass for the selected script, enter the fully-qualified class name of your custom helper superclass in the Helper Superclass field. Note that your helper superclass must extend RationalTestScript.

Results

Note: If you change your superclass and then want to reset it back to RationalTestScript later on, you can either type RationalTestScript in the superclass field or just clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

Script helper superclass/base class

A helper superclass or base class is an optional, user-written class that provides override support for base-level methods, in particular, the event handler methods.

See `com.rational.test.ft.script` in the **Rational® Functional TesterHCL OneTest™ UI API Reference** for more information on superclass. See `Rational.Test.Ft.Script` in the **Rational® Functional TesterHCL OneTest™ UI API Reference** for more information on base class.

Creating a script helper superclass

By default, all Functional Test scripts extend the `RationalTestScript` class, and thereby inherit a number of methods (such as callScript). If you are an advanced user, you might want to create your own helper superclass, which extends `RationalTestScript` and adds additional methods or overrides the methods from `RationalTestScript`.

Before you begin

About this task

To create a script helper superclass for a script:

1. Click **File > New > Helper Superclass** or click the **View Menu** button next to the **New** button on the product toolbar and click **Helper Superclass**.
   The Create Script Helper Superclass dialog box opens.
2. In the folder field, either enter the appropriate path to the folder or use the navigation tools (**Home**, **Back**, and **Go Into**) to select the path that contains the project for which you want to create a helper superclass.
3. Select a project name in the project list.
4. Enter a class name in the **Script name** field.
5. Click **Finish**.
   Rational® Functional TesterHCL OneTest™ UI creates a new script in the Java™ Editor that you can use to manually enter Java™ code. The cursor appears at the top of the script.
6. Enter the methods and member variables you want to make available to the script.

Results

Note: When you create a script helper superclass, you can override base-level functionality from the `RationalTestScript` class.

What to do next

Changing the default script helper superclass

By default, all Functional Test scripts extend the RationalTestScript class, and thereby inherit a number of methods (such as callScript). Advanced users may want to create their own helper superclass which extends RationalTestScript and can add additional methods or override the methods from RationalTestScript.

About this task

You can specify a helper superclass that Rational® Functional TesterHCL OneTest™ UI will use whenever you create or record a script in your project. This default superclass is specified in the Functional Test Project Properties page. You can also specify a helper superclass for an individual script in the Functional Test Script Properties Page. Once a script has been created, it retains the reference to the default superclass as its own helper superclass.

To change the default script helper superclass for a project:

1. Select a project in the Projects view.
2. Right-click and select **Properties**.
3. Click **Functional Test Project**.

   The Functional Test Project Properties page opens. The helper superclass listed here will be used by default for all new scripts created or recorded in this project.

   Note: This is a user-specific preference and will not be shared by other users of this project.
4. To change the default superclass for the selected project, enter the fully-qualified class name of your custom helper superclass in the Default Script Helper Superclass field. Note that your helper superclass must extend RationalTestScript.

Results

Note: If you change your superclass and then want to reset it back to RationalTestScript later on, you can either type RationalTestScript in the superclass field or just clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

To change the script helper superclass for an individual script:

1. Select a script in the Projects view.
2. Right-click and select **Properties**.
3. Click **Functional Test Script**.

   The Functional Test Script Properties Page opens. The helper superclass listed here will be used for the script you selected.
4. To change the default superclass for the selected script, enter the fully-qualified class name of your custom helper superclass in the Helper Superclass field. Note that your helper superclass must extend RationalTestScript.

Results

Note: If you change your superclass and then want to reset it back to RationalTestScript later on, you can either type RationalTestScript in the superclass field or just clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

Script support functions

When you record or edit a functional test script, you can insert codes to add features into the test script by using the tabs in the **Script Support Functions** dialog. The following table lists the tab names and the features associated with each of the tabs:

Tab name	You can...
Call Script	Insert a statement to call another test script.
Log Entry	Insert a log message into the test script. During playback, this message is displayed in the log.
Timer	Insert or stop a timer in the current script. Timers remain in the running state until you stop them explicitly or close Rational® Functional TesterHCL OneTest™ UI.
Sleep	Insert a sleep command into the test script to delay the script for the time that you specify.
Comment	Insert a comment into the test script.
Clipboard	Insert system clipboard commands into the test script.

Note: You can open the **Script Support Functions** dialog in one of the following ways:

- You can click the **Insert Script Support Commands** button on the **Recording Monitor** toolbar when you record the test script.
- You can click the **Insert Recording into Active Functional Test Script** button on the **Functional Test** toolbar, and then click the **Insert Script Support Commands** button on the **Recording Monitor** toolbar when you edit the test script.

The following links provide more information:

- Calling a script from a functional test script on page 891
- Inserting a log message into a functional test script on page 892
- Using timers with functional test scripts on page 893
- Setting delays and sleep states for functional test script playback on page 894
- Inserting comments into a functional test script on page 895
- Inserting clipboard commands into a functional test script on page 895

Calling a script from a functional test script

While recording or editing a functional test script, you can insert a call to a previously recorded script. This lets you avoid repeatedly recording similar actions on the application-under-test by taking advantage of scripts that already exist.

Before you begin

About this task

To call a script from a functional test script:

1. If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar.

   If editing:

   a. Position the pointer in the script where you want to place the callScript command.
   b. Click the **Insert Recording into Active Functional Test Script** button on the product toolbar.
   c. Click the **Insert Script Support Commands** button on the Recording Monitor toolbar.
2. Click the Call Script tab in the Script Support Functions dialog box.
3. In the **Script Name** field, select from the list the name of the script you want to call or enter the name.
4. In the **dataset Iterator Count** field, do one of the following:
   Choose from:
   - Type or select the number of records in the dataset.
   - Select **Iterate Until Done** to access all records in the dataset.
   - Select **Use Current Record** to use the same record across the call script.
5. Click **Insert Code**.

   Rational® Functional TesterHCL OneTest™ UI inserts the `callScript ("scriptname")` code at the cursor location, where *scriptname* is the name you selected in the **Script Name** field.
6. Click **Close** to remove the Script Support Functions dialog box from the screen.

Note: You can also insert one or more callScript commands from the Functional Test Projects view Solution Explorer.

What to do next

Inserting a log message into a functional test script

You can insert a log message into a functional test script and indicate whether it is a message, a warning, or an error. During playback, Rational® Functional TesterHCL OneTest™ UI inserts this information into the log.

About this task

To insert a log message into a functional test script:

1. If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar.

   If editing:

   - Position the pointer in the script where you want the log message.
   - Click the **Insert Recording into Active Functional Test Script** button on the product toolbar.
   - Click the **Insert Script Support Commands** button on the Recording Monitor toolbar.

2. Click the Log Entry tab on page 1788 in the Script Support Functions dialog box.
3. In the **Message to write to the log** field, enter the message text.
4. In the **Result** section, select either **Information**, **Warning**, or **Error**.
5. Click **Insert Code**.

   Rational® Functional TesterHCL OneTest™ UI inserts code in the script based on the option you selected in the **Result** section, where `message` is the text you entered:

   ```
   logInfo(" message")
   ```

   ```
   logWarning(" message")
   ```

   ```
   logError(" message")
   ```

   Note: You can pass image as an second argument to these methods. For example, when you want to get a screen capture along with an error message, you can use the second argument as shown in the following example:

   ```
   logError("error message here", getRootTestObject().getScreenSnapshot());
   ```

6. Click **Close** to remove the Script Support Functions dialog box from the screen.

Using timers with functional test scripts

You can insert any number of timers with different names into the same script to measure the time it takes to perform a variety of separate tasks. You can nest timers within other timers (starting and stopping the second timer before stopping the first timer), and you can overlap timers (stopping the second timer after stopping the first timer).

About this task

However, you should stop a timer before starting that same timer again. If you start the same timer again, Rational® Functional TesterHCL OneTest™ UI changes the starting time. When you stop a timer, Rational® Functional TesterHCL OneTest™ UI writes a message to the log that indicates the time elapsed from when the timer started. If you stop the same timer multiple times, Rational® Functional TesterHCL OneTest™ UI does not restart the timer. You should call timerStart if you want to restart the timer.

When you play back a script that includes timers, you can view the elapsed time in the log.

To insert a timer while recording or editing a script:

1. If recording, click the **Insert Script Support Commands** button on the Recording toolbar.

   If editing:

a. Position the pointer in the script where you want to place the timer.
b. Click the **Insert Recording into Active Functional Test Script** button  on the product toolbar.
c. Click the **Insert Script Support Commands** button  on the Recording toolbar.

2. Click the **Timer** tab on page 1818 in the Script Support Functions dialog box.
3. In the **Start Timer: Name** field, type a timer name. If you start more than one timer, make sure you give each timer a different name.
4. Click **Insert Code**.

   Rational® Functional TesterHCL OneTest™ UI inserts the `timerStart("name")`code at the cursor location in the script where *name* is the name you entered in **Start Timer: Name** field.

5. Perform the activity you want to time.
6. Immediately after the timed activity, stop the timer:

   a. Click the **Insert Script Support Commands** button  on the Recording toolbar.

   b. Click the **Timer** tab in the Script Support Functions dialog box.

   c. In the **Stop Timer: Timers** field, select from the list the timer that you want to stop.

      If you do not see the timer name in the list, type the name in the combo box.

   d. Click **Insert Code**.

      Rational® Functional TesterHCL OneTest™ UI inserts the `timerStop("name")` code at the cursor location in the script where *name* is the name you selected in **Stop Timer: Timers** field.

      Note: Do not insert a `timerStop` statement before the corresponding `timerStart` statement.

Setting delays and sleep states for functional test script playback

You can insert a sleep command into your functional test script to delay the script for the amount of time you specify.

About this task

To insert a sleep code while recording or editing a script:

1. If recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar.

   If editing:

   a. Position the pointer in the script where you want to insert the Sleep command.
   b. On the product toolbar, click the **Insert Recording into Active Functional Test Script** button  .
   c. On the Recording Monitor toolbar, click the **Insert Script Support Commands** button  .

2. In the Script Support Functions dialog box, click the Sleep tab on page 1818.

3. In the **(seconds)** field, enter the time in seconds you want to delay the Functional Test script execution. You can use floating point numbers for the seconds; for example, `sleep(0.1)` waits for 1/10 of a second.
4. Click **Insert Code**.

   Rational® Functional TesterHCL OneTest™ UI inserts the `sleep(seconds)` code at the cursor location in the script where *seconds* is the time you entered in the **(seconds)** field.

Inserting comments into a functional test script

During recording or editing, you can insert lines of comment text into a Functional Test script. Comments are helpful for documenting and editing scripts.

About this task

To insert a comment into a script during recording or editing:

1. If recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar.
2. If editing:
   a. Position the pointer in the script where you want to insert the comment.
   b. On the product toolbar, click the **Insert Recording into Active Functional Test Script** button  .
   c. On the Recording Monitor toolbar, click the **Insert Script Support Commands** button  .
3. In the Script Support Functions dialog box, click the Comment tab on page 1743.
4. In the **Comment to add to the script** field, type the comment. Rational® Functional TesterHCL OneTest™ UI does not automatically wrap the text. Put returns after each line.
5. Click **Insert Code**.

   Rational® Functional TesterHCL OneTest™ UI inserts the text with the appropriate comment delimiter (//) preceding each line.

Inserting clipboard commands into a functional test script

During recording or editing, you can insert system clipboard commands into a functional test script. You can also insert into a functional test script a verification point test command against the active content in the system clipboard.

About this task

To insert a clipboard command into a script during recording or editing:

1. If recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar.
2. If editing:
   a. Position the pointer in the script where you want to insert the comment.
   b. On the functional testing toolbar, click the **Insert Recording into Active Functional Test Script** button  .
   c. On the Recording Monitor toolbar, click the **Insert Script Support Commands** button  .

3. In the Script Support Functions dialog box, click the Clipboard tab on page 1741.
4. To insert the clipboard verification point test command into the script, select the **Verification Point** tab and perform the following:

    a. Type the verification point name in the **VP Name** field.

    b. Click the **Convert Value to Regular Expression** button to convert the system clipboard value to a regular expression pattern.
    This is matched at run time against the system clipboard contents.

    c. Click the **Evaluate Regular Expression** button to evaluate the current pattern against the system clipboard contents.
    Result
    In the Evaluate Regular Expression on page 1810 dialog box, the **Pattern** and **Match Against Value** fields contain the current value. To try an expression, change the value in the **Pattern** field and click the **Evaluate** button. The **Result** indicates whether the expression matched. Click **OK**.

    d. Click the **Insert Code** button to insert the clipboard verification point command into the functional test script.

5. You can also have clipboard text automatically pasted into an input field during test runs. To assign the clipboard text to a script variable, perform the following: and perform the following:
    a. Select the **Assign Text** tab
    b. Type the script **Variable Name** to which you want to assign the clipboard text.
    c. Select the **Precede variable assignment with type declaration** check box to precede the variable name with a string type declaration.
    d. Click the **Insert Code** button to insert in the functional test script the command for assigning the clipboard content into a local variable.
6. To update the contents of the system clipboard, select the **Set Text** tab and perform the following steps:
    a. In the **Set clipboard text to the following value** field, type the value for the clipboard.
    b. Click the **Insert Code** button to insert into a functional test script the command for setting the content of the clipboard to the supplied value.

Starting your test applications

When recording tests on your application, it is best to have Rational® Functional TesterHCL OneTest™ UI start the application during recording. This makes playing back the tests more reliable because the application configuration information is used. Rational® Functional TesterHCL OneTest™ UI can open specified Java applications, HTML pages in your browser, or run executable applications.

1. During recording, click **Start Application** on the Recording toolbar.
    The Start Application dialog box opens.
2. Click the arrow in the **Application Name** field to see the list of applications that you can test.

- Java™: *Application name* - java
- HTML: *Application name* - html
- Executable or batch files *Application name* - executable

3. Select the application to open, and click **OK**.
   The dialog box closes and your application opens.Rational® Functional TesterHCL OneTest™ UI inserts a test line to start the application.

   Note: You can configure your own applications so that the **Application Name** list includes the applications and you can start them using this dialog box. You add your applications by clicking the **Edit** button. For information on editing or adding application information see *Configuring Applications for Testing* related topic.

   Rational® Functional TesterHCL OneTest™ UI comes with several sample applications that the **Application Name** list includes. For example, "ClassicsJavaA - java" is used in the Rational® Functional TesterHCL OneTest™ UI tutorial.

Renaming a test asset

You can rename simplified test scripts, Java test scripts, test object maps, or other files in a project.

About this task

When you rename a script, Rational® Functional TesterHCL OneTest™ UI renames all its related files, such as the helper script files, the private object map, and any verification point files. When you rename a test object map, Rational® Functional TesterHCL OneTest™ UI updates associated scripts with the new name.

Note:

- You need to change any callScript commands in scripts that reference the old script name; otherwise, Rational® Functional TesterHCL OneTest™ UI logs an error when you run those scripts.
- For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, a Rename command is available in the Navigator view that is part of the Eclipse Workbench. This Rename command only renames an individual file, not the collection of files that makes up a Functional Test script. Therefore, do not use the Rename command in the Navigator view to modify any Functional Test project assets.

1. From the Functional Test Projects view, right-click a script or test asset.
2. Click **Rename**.
3. Type the new test asset name in the **New name** box, and click **Finish**.

   Note: If you rename a simplified script, the associated Java script is also renamed.

4. You can click a script or test asset, press **F2**, type the new test asset name, and press **Enter** or you can do the following:

a. From the Solution Explorer, right-click a script or test asset.
b. Click **Rename**.
c. From the Solution Explorer, type the new test asset name and press **Enter**.

Saving test scripts and files

You can save a functional test script or file in several ways: save the current test script or file, save all test scripts and files, save a functional test script or file with another name in a different location.

- To save the current test script of file, click **File > Save *name*** from the Rational® Functional TesterHCL OneTest™ UI menu or click the **Save** button on the product toolbar.
- To save all the test scripts and files, click **File > Save All**.
- To save a test script with another name, click **File > Save Script *scriptname As*** from the product menu. **File > Save *Scriptname* As** from the product menu.

Saving a test script with another name

1. Click **File > Save Script *scriptname* As File > Save *Scriptname* As** from the product menu to open the Save Functional Test Script As on page 1813 dialog box.
2. In the **Select a folder** field, use the navigation buttons (**Home**, **Back**, and **Go Into**) **Save in** field, use the navigation buttons (**Save In** arrow, **Current Folder**, **Up One Level**) to select the appropriate path to the folder you want to use.
3. In the **Script name** field, enter a name that conforms to the Java™ naming conventions for the new script. **File name** field, enter a name that conforms to class name requirements for the new script.

   Note: The script name is appended to the project path. The project path and script name together should not exceed 230 characters.

4. Click **Finish Save**.
   Rational® Functional TesterHCL OneTest™ UI saves the script and all its related files, such as the helper script files, the private object map, and any verification point files using the new name.

Saving a file with another name

About this task

Do not use this procedure to save a primary script class. A script is a collection of files that include the primary script class, verification point files, helper class, and possibly a private object map. If you save only the primary script class by using this procedure, the other files included in the script are not renamed and Rational® Functional TesterHCL OneTest™ UI cannot play back the resulting file. To save a script and its related files, use the Save Script As dialog box

1. Click **File > Save As File > Save *file* As** in the product menu to open the Save As dialog box.
2. In the **Enter or select the parent folder** field, use the navigation buttons to select the appropriate path to the folder you want to use.

3. In the **File name** field, enter the name for the new file.
4. Click **OK Save**.
   Unlike the Save Script As dialog box, the Save As dialog box saves only the current file and not any related files.

Deleting a functional test script

You can delete a functional test script from the Projects view or the Solution Explorer. Deleting a script cannot be undone.

1. In the Projects view Solution Explorer, right-click the script you want to delete.
2. Click **Delete**.
   Result

   Rational® Functional TesterHCL OneTest™ UI displays a message asking you to confirm the deletion.
3. Select **Yes**.

Tips for recording functional tests

Following are some tips on ways around some potential recording issues.

Enabling your JREs and Web browsers

You must enable your JREs for Java™ testing and your browsers for HTML testing. If your JREs or browsers are not enabled, the Recording Monitor is blank when you try to record against a Java™ or HTML application. For this reason, leave the Record Monitor in view while recording. If you see this symptom, you need to run the enablers. For information, see topics *Enabling Java Environments* and *Enabling Web Browsers*.

Under certain conditions, you can start recording a script to test an application without having to prepare your test environment. The components involved in running the test application and recording the script are enabled automatically when you start recording. See Recording scripts without preparing the environment on page 774 for more information.

Enabling and testing Eclipse shells and Eclipse RCP applications

Rational® Functional TesterHCL OneTest™ UI can be used to test Eclipse shell extensions. You must enable Eclipse, using the Eclipse Platforms tab of the Rational® Functional TesterHCL OneTest™ UI enabler. If your Eclipse shell is not enabled, you will be able to tell because the Recording Monitor will be blank when you try to record against it. For this reason, leave the Recording Monitor in view while recording. If you see this symptom, you need to run the enabler. See the Eclipse Platforms Tab topic for information about enabling an Eclipse-based application. Note that Rational® Functional TesterHCL OneTest™ UI cannot be used to test an Eclipse in which it itself is running (one in which the Functional Test perspective has been loaded). If you have the Functional Test perspective loaded and you then close it, you will need to close and restart Eclipse itself before you can test.

You can test an instance of the Eclipse shell running from the same installation as Rational® Functional TesterHCL OneTest™ UI as long as it uses a separate workspace. You can use the -data parameter to the eclipse.exe command line to specify the workspace. See the Eclipse documentation for details.

Dynamic enablement of Windows®, and .Net applications

There is no enabler for Windows®, and .Net applications. Instead, Rational® Functional TesterHCL OneTest™ UI can dynamically enable these applications. During recording, you will notice a delay when you first use the mouse to click a control, or type keystrokes into one of these types of applications. During this delay the mouse will freeze. This delay is caused by the dynamic enablement.

Once an application is dynamically enabled, it stays enabled until it is closed.

If you use one of the object selection wizards (for example while recording a verification point) on an application before it is dynamically enabled, the objects in the application will not be highlighted. After the object is selected, the application will be dynamically enabled by the wizard. After it is dynamically enabled, the highlighting mechanism will work as usual.

Java™ script naming conventions

Scripts created in Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration must follow Java™ Class naming conventions. For example, script names cannot contain spaces or non-alphanumeric characters, nor can a script name begin with a number. While it is not mandatory to do so, it is traditional to begin a Java™ class name with a capital letter. Names of methods (such as verification point helper methods) have the same restrictions, but it is traditional to begin a Java™ method with a lower case letter. Finally, when a class or method name is made up of multiple words, it is traditional to capitalize the additional words. For example, you might use ApplicationMenuTest for a class name and validateFileMenu() for a method name.

VB.Net script naming conventions

Scripts created in Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration must follow VB.Net Class naming conventions. For example, script names cannot contain spaces or non-alphanumeric characters, nor can a script name begin with a number. While it is not mandatory to do so, it is traditional to begin a VB.Net class name with a capital letter. Names of methods (such as verification point helper methods) have the same restrictions, but it is traditional to begin a VB.Net method with an upper case letter. Finally, when a class or method name is made up of multiple words, it is traditional to capitalize the additional words. For example, you might use ApplicationMenuTest for a class name and ValidateFileMenu() for a method name.

Changing state of an application while recording verification points

When you have the recorder paused to create a verification point, be aware that if you change the state of the application in any way, it may affect your script. If you change the state of the application while the recorder is paused, actions recorded after this could prevent the script from playing back because the application is in the wrong state to play back the actions. Before you start to create the verification point, be sure to put the application in whatever state you need it to be in. For example, if you need to do actions in the user interface to locate the object you want to test, put the application in that state before you start the verification point.

Note: Rational® Functional TesterHCL OneTest™ UI offers an object selection method to access objects that may be difficult to select. In the Select an Object page of the Verification Point and Action Wizard, there is a delayed object selector you can use to pause the recorder while you access an object in the application. Actions done while the delay is in effect will not be recorded.

Changing state of an application while recording a data-driven test

When you have the recorder paused to create a data-driven test, be aware that if you change the state of the application in any way, it may affect your script. If you change the state of the application while the recorder is paused, actions recorded after this could prevent the script from playing back because the application is in the wrong state to play back the actions. Before you start to create the verification point, be sure to put the application in whatever state you need it to be in. For example, if you need to do actions in the user interface to locate the object you want to test, put the application in that state before you start the verification point. You may find it is convenient to take advantage of the paused recorder to change the data contents of the controls you are going to data drive. This way, the recorder will not record redundant actions to set the data contents of the controls.

Selecting objects shortcut

Here's a neat shortcut. While recording, you can drag the **Verification Point and Action Wizard** button on the Record User Actions toolbar to immediately start selecting an object in your application. This is a shortcut for selecting it from the Select an Object Page of the Verification Point and Action Wizard. You will then be in the wizard after you select the object.

Recording scrolling actions

Scroll actions are commonly ignored in both Java™ and HTML recording. At playback, subitems are auto-scrolled into view before being acted upon, making the scroll actions rather irrelevant. In some cases Functional Test still records scroll actions when they are host-independent (in the case of JFC applications) to keep the recording as close as possible to the actual actions performed, though Rational® Functional TesterHCL OneTest™ UI still auto-scrolls as necessary during playback to ensure that things work as expected.

Scroll actions will not cause failures during playback, since Functional Test auto-scrolls anyway and will ignore the normal out-of bounds and scrollbar-not-visible type of errors that can occur with scroll actions. Scroll actions tend to fail fairly commonly with cross-platform script execution, so ignoring scroll failures improves the cross-platform nature of scripts.

Hover feature

When you record actions in an HTML application, you can use this hover feature to move the mouse to a particular place during playback. This is particularly useful for clicking on menus or links in HTML testing. To use the feature, move the mouse pointer onto the object for which you want to record a hover. Press and then release the **Shift** key to record the hover. This will cause the recorder to insert a hover() method in the script. At playback, the mouse will then be able to activate links and menus by hovering on them, instead of clicking. You can use multiple hover actions to

support a cascading menu. The Record Monitor will give you a warning message if you click where there is no object or if hover is not supported for an object.

Maximizing the script window

To maximize the script window (the Java™ Editor), double-click the tab with the script name. This makes it easier to edit the code. Double-click the tab again to restore Rational® Functional TesterHCL OneTest™ UI to normal.

Recording a script

To start recording scripts against your applications, you must first configure your test environments, configure your applications, and create a project. Under certain conditions, you can start recording scripts without preparing your functional test environment.

About this task

If you haven't done these configuration tasks, see Getting Started with Rational® Functional TesterHCL OneTest™ UI for the necessary steps.

When you record a script, Rational® Functional TesterHCL OneTest™ UI records any user actions against your application, such as keystrokes and mouse clicks. You can also insert verification points to test data or properties of any objects in your application. During recording, the verification point captures object information and stores it in a baseline file. Then during playback, the verification point captures the object information and compares it to the baseline.

Note: All Functional Test scripts use a default helper base class. You can create your own helper base class if you want to add additional methods or override the methods within RationalTestScript. For more information, see Changing the Default Script Helper Base Class for a Script on page 887.

To record a script:

1. Optionally, you can first set any recording options you may need. Click **Tools > Options** to access the Rational® Functional TesterHCL OneTest™ UI options. In the folder hierarchy, click the **Rational Functional TesterHCL OneTest UI** folder icon to open the options. Set any recording options. Close the options when you are done.
2. Click the **Record a Functional Test Script** button or click **File > New > Add Script Using Recorder**. The Add New Item dialog box opens.
3. In the Add New Item dialog box, select the project you want the script to be part of. Type a name for the script.
4. Click **Open**.

   The Select Script Assets page on page 1834 appears. Rational® Functional TesterHCL OneTest™ UI will create a local object map for your script by default. If you want to use a different test object map on page 1838, helper base class on page 1829, or test dataset on page 938, select them from the Select Script Assets page. You can also set the dataset record selection order or change the dataset associated with the script.

5. Click **Finish** to begin recording.

   The Recording Monitor opens and the recording starts. Click the **Display Help** icon on the Recording toolbar in the monitor for information on the toolbar buttons and how the monitor works.

6. On the **Recording** toolbar, click the **Start Application** button to start your test application. See Starting Your Test Applications on page 896 for more information. (If your application is already running, you don't need to do this step.)
7. Perform any actions in the application.

   Note: To record the action of moving the mouse over a link that has a tooltip, move the mouse over the link so that the tooltip is displayed, and press Shift. This notifies the recorder to capture the action in the script.

8. If you want to record a verification point, locate the object in your application you want to test, and click anywhere in the application window or dialog box. Next, click the **Insert Verification Point or Action Command** button. Click the **Help** button in the Verification Point and Action Wizard while creating the verification point for more information on the Verification Point and Action Wizard, or see Creating a Properties Verification Point on page 904 for an example of how to create one type of verification point.
9. If you want to insert any features into the script, such as a call script command, log entry, timer, script delay command, or comment, click the **Insert Script Support Commands** button. Click the **Help** button in the Script Support Functions on page 890 dialog box for information on script support functions.
10. Close your application, if you want closing the application to be part of the script.
11. When you are finished recording, click the **Stop Recording** button.

    Note:
    - For recording tips and troubleshooting information, see Recording Troubleshooting and Tips on page 899.

Working with functional test object maps in the Visual Studio IDE

This section describes how you can use test object maps to facilitate your functional testing effort. The test object map is a static view that describes the known test objects in the application-under-test.

Exemple

You can refer to the following topics about test object maps:

- Creating a new test object map
- Merging multiple test object maps
- Adding objects to a test object map
- New test objects in an object map
- Adding a test object to an object map and a script
- Editing object properties

- Displaying associated scripts
- Renewing a name in associated script or scripts
- Resetting object names to the default names in associated scripts
- Adding test objects to a script
- Adding test object descriptions
- Editing test object descriptions
- Deleting test objects from an object map
- Unifying two test objects
- Maintaining a test object map
- Updating recognition properties
- Deleting a test object map
- Locating a test object in the application
- Searching for objects in a test object map
- Regular expressions
- Object library

Verification points

Verification points verify whether an expected behavior occurred during a test playback, or they verify whether there is a change in the state of a control or an object. When you create a verification point, you capture information about a control or an object in the application to establish this information as a baseline for comparison during the playback.

You can use verification points to identify possible defects when you upgrade an application to the next version. Rational® Functional TesterHCL OneTest™ UI reports an error if the expected behavior does not occur during the playback.

The following links provide more information:

- Creating properties verification point on page 904
- Creating a data verification point on page 907
- Editing verification point data on page 909
- Selecting objects and data value options for data verification points on page 911
- Creating an image verification point on page 914
- Verification Point Editor on page 917
- Verification point comparator on page 923
- Comparing and updating verification point data using the Comparator on page 933
- Inserting verification points into the script using the application visuals on page 934

Creating properties verification point

Use a Properties verification point to test properties of an object in your application. When you record the verification point, a baseline of the data is created. Then every time you play back the script, the data will be compared to see whether any changes have occurred, either intentionally or unintentionally. This is useful for identifying possible

defects. You can create a verification point while recording a script or you can insert a verification point anytime in the script.

Before you begin

When you create a verification point you can use a dataset reference instead of a literal value to supply variable data to make your tests more realistic. You can use a dataset reference for a string, a number, a color, or a boolean instead of a literal value in a properties verification point. You cannot use a dataset reference instead of a literal for more complex objects such as a font, a point, or a rectangle for a properties verification point.

Note: Avoid creating properties verification point on a higher level control in SAP applications if it contains multiple children controls.

Prerequisites:

- The test application is started
- If you are inserting a verification point to an existing script, open the script and place the cursor at the point in the script to insert the verification point.

1. Open the Verification Point and Action wizard.
   - If you are creating a verification point while recording, click the **Insert Verification Point or Action Command** button on the Recording Monitor toolbar.
   - If you are inserting a verification point on a script, click the **Insert Verification Point into Active Functional Test Script** button on the product toolbar.
2. On the **Select an Object** page of the Verification Point and Action wizard, use any one of the object selection method to select the object in your application.
   For more information, see the related topic on *Selecting an Object Page*.

   Note: By default, the **After selecting an object advance to next page** check box on the Select an Object page is selected. After you select an object, the next page in the Verification Point and Action wizard appears. If you clear the **After selecting an object advance to next page** check box, after you select an object, you must click **Next** to proceed to the next page.

3. On the **Select an Action** page, click the **Perform Properties Verification Point** option and click **Next**.
4. On the Insert Properties Verification Point Command page, perform the following steps:

   a. If you want to include the children of the selected object for properties verification point, click **Immediate** or **All** in the **Include Children** field.

   b. In the **Verification Point Name** field, accept the default suggestion, or type a new name.
   This name must follow standard Java naming conventions. The default name is based on the name of the object and the data value you chose to test.

   c. Select the **Use standard properties** check box if required.

Standard properties are properties available across platforms and browsers. Nonstandard properties may include platform-specific properties.

d. Use the **Include Retry Parameters** to set a retry time for a verification point during playback to check for the existence of the verification point in the application. The retry option is useful when playback does not find immediately the verification point in your application. To set a retry time, either use the default, or set a time of your choice. **Maximum Retry Time** is the maximum number of seconds that the functional test will wait for a verification point to become available for retesting. **Retry Interval** is the number of seconds between times that the functional test will check for the verification point during the wait period.

e. Click **Next**.

5. On the Verification Point Data page, edit the required properties data.
   The test object properties and their values are displayed in a tree table format. You can edit which properties get tested in the **Property** column, and can edit the property values themselves in the **Value** column.
   - To edit the list of object properties that gets tested during playback, use the check box beside each property. Checked properties are tested each time you play back a script with this verification point. Use the **Check All** or **Uncheck All** buttons to select or clear all the properties in the list.
   - To edit a property value, double-click the Value cell and edit it.
6. **Optional:** To use a dataset reference instead of a literal value for a property verification point:
   a. In the **Property** column, select a property, right-click, and then click **Convert Value to dataset Reference**. The dataset Reference Converter dialog box opens.
   b. Type a new name for the dataset variable or click the **dataset Variable** arrow to select the variable that you want the verification point to reference in the dataset
   c. Optionally, select the **Add value to new record in dataset** check box to add the value of the verification point to a new record (row) in the dataset.
   d. Click **OK**.
7. Click **Finish**.
   Result

Notes:

- A warning is displayed if you click **Finish** without selecting any properties for the verification point in the Verification Point Data page.
- You can also create a verification point by manually scripting it. For more information, see *Adding manual and dynamic verification points* topic. Also see the vpManual() and vpDynamic() methods in the API reference topics. Your script can access the same information as the verification points. See the TestObject getProperty() and getTestData() methods.
- You can change a property value to a regular expression or numeric range, or change one of them back to its original property value, using the Verification Point Editor. For more information, see *Replacing an Exact-Match Property with a Pattern* topic
- While inserting the verification point without using the Recorder in the script, the test object is not inserted in the script. You must manually insert the test object for which you



are creating a verification point. For example: `.performTest( Screen_imageVP() );` script is inserted when you insert an image verification point without using the Recorder. You must include `RootTestObject.getScreenTestObject()` to the script. The script must be `RootTestObject.getScreenTestObject().performTest( Screen_imageVP() );` for the verification point to work.

Creating a data verification point

Use a data verification point to test data that is displayed in your application. When you record the verification point, a baseline of the data is created. Then every time you play back the script, the data is compared to see whether any changes have occurred, either intentionally or unintentionally. This is useful for identifying possible defects. You can create a verification point while recording a script or you can insert a verification point anytime in the script.

Before you begin

You must have completed the following tasks:

- Start the application which you want to test.
- When you want to insert a verification point to an existing script, open the script and place the cursor at that point in the script where you want to insert the verification point.

About this task

You can either create a data verification point when you record a script or insert a data verification point after you record a script. The data verification point captures the data displayed for the selected objects. For the objects that contain a script tag, the data verification point also captures the JavaScript code. If you want to record or run the test without capturing the JavaScript code, you can add the flag *ft.html.text.skipscripttag* and set it to *true* in `ivory.properties` file, which is located in the folder `<install_directory>\Functional Tester\bin\`.

You can test the following types of data in your application:

- List data
- Menu hierarchy
- The state of a checkbox or a toggle button in your application
- Table data
- Data that is displayed in a DataGrid control
- Data that is displayed in a DataGridView control
- Data that is displayed in a ToolStrip control
- Textual data
- Tree hierarchy

Tip: When you create a verification point, you can use a dataset reference instead of a literal value to supply variable data to make your tests more realistic.

1. Open the Verification Point and Action wizard.
   - If you are creating a verification point while recording, click the **Insert Verification Point or Action Command** button on the Recording Monitor toolbar.
   - If you are inserting a verification point on a script, click the **Insert Verification Point into Active Functional Test Script** button on the product toolbar.
2. On the **Select an Object** page of the Verification Point and Action wizard, use the Object Finder to select the data that you want to test.
   For more information, See Selecting objects and data value options for data verification points on page 911.

   Tip: If the **After selecting an object advance to next page** check box on the Select an Object page is selected, the next page of the wizard is displayed after you select the object. If this check box is cleared, click **Next** to proceed to the next page.

3. On the **Select an Action** page, select the **Perform Data Verification Point** option and click **Next**.
4. On the **Insert Verification Point Data Command** page, perform these steps:

   a. Click one of the available options in the **Data Value** field and click **Next**.
   For information about the data value options and description, see Selecting objects and data value options for data verification points on page 911.

   Notes:
   - The options shown in the **Data Value** field depends on information provided by the proxy of the object. Values other than those described in the related topic might be listed in the Data Value field.
   - The Visible Table Contents option for DataVP is not available for OLAPDataGrid.

   b. In the **Verification Point Name** field, accept the default suggestion, or type a new name.
   This name must follow standard Java naming conventions. The default name is based on the name of the object and the data value that you chose to test.

   c. To verify that verification point exists in the application, use the **Include Retry Parameters** to set a retry time for a verification point during playback. The retry option is useful when playback does not find immediately the verification point in your application. To set a retry time, either use the default, or set a time of your choice. Specify these settings:
   - **Maximum Retry Time**: The maximum number of seconds that the functional test waits for a verification point to become available for retesting.
   - **Retry Interval**: The number of seconds between times that the functional test checks for the verification point during the wait period.

For more information, see Editing verification point data on page 909.

d. Click **Next**.

5. On the **Verification Point Data** page, edit the data if required.
6. **Optional:** To use a dataset reference instead of a literal value for a data verification point:
   a. Right-click a property, and then click **Convert Value to dataset Reference**. The dataset Reference Converter dialog box opens.
   b. Type a new name for the dataset variable or click the **dataset Variable** arrow to select the variable for the verification point to reference in the dataset.
   c. **Optional:** Select the **Add value to new record in dataset** check box to add the value of the verification point to a new record (row) in the dataset.
   d. Click **OK** and close the text box.
7. Click **Finish**.

   Result

   The verification point is added to the script. You can edit the verification point any time by using the Verification Point Editor. For more information, See Verification Point Editor on page 917.

Notes:

- You can also create a verification point by manually scripting it. For information, see the *Adding manual and dynamic verification points* topic. Also see the vpManual() and vpDynamic() methods in the API reference topics. Your script can access the same information as the verification points. See the TestObject getProperty() and getTestData() methods
- When you create a verification point without using the Recorder in the script, the test object is not inserted in the script. You manually insert the test object for which you are creating a verification point. For example: this script is included when you insert an image verification point without using the Recorder `.performTest( Screen_imageVP() );`. For the verification point to work, include `RootTestObject.getScreenTestObject()` in the script. This is the script for the verification point `RootTestObject.getScreenTestObject().performTest( Screen_imageVP() );`

.

Editing verification point data

You can edit verification point data while creating or inserting a verification point in the Verification Point Data page wizard. After you create the verification point, you can edit the data in the Verification Point Editor.

Editing data verification points for listing elements

When you create a data verification point and choose the **List Elements** test, the data is displayed in a list format in the main data area of the Verification Point Editor and the Verification Point Data wizard page. The list displays the same information as the list in your application, in the same order.

- To edit the list of items to test during playback, mark the check box beside each item. Checked items are tested each time you play back a script with this verification point. Click the **Check All** or **Uncheck All** to select or clear all of the items in the list.
- To change a value in the list, double-click the list item and change the value.
- To insert items into the list, click the **Insert** toolbar icon  to insert a blank line. Type the new list item in the line.

Editing data verification points for menu hierarchy tests

When you create a data verification point and choose the **Menu Hierarchy** or **Menu Hierarchy with Properties** test, the menus are displayed in a tree format in the main data area of the Verification Point Editor and the Verification Point Data wizard page. The tree displays the entire menu hierarchy of your application, or one top-level menu and its sub-items, depending on how you recorded the verification point. If you chose the whole menu bar, each top-level menu is shown from top to bottom in the tree in order that they occur from left to right in the menu bar. Each menu item is shown under its top-level menu. Use the plus and minus signs to open and close the list for each top-level menu.

- To edit the list of menu items for testing during playback, use the check box beside each item. Checked items are tested each time you play back a script that includes this verification point. Click **Check All** or **Uncheck All** to select or clear all the items in the list.
- To edit a menu item properties:
    1. Double-click the menu item in the tree. The menu item properties dialog box displays the properties in a grid.
        - To edit the actual values, double-click the value in the **Value** column of the grid and change the value.
        - To edit the detailed properties of the menu item, double-click **Masked Property Sheet**, and edit the required values. The Masked Property Sheet value is listed only for the Menu Hierarchy with Properties test.
    2. Close the menu item properties dialog box.

Editing data verification point for a table

When you create a data verification point and choose the Table Contents or Selected Table Cells test, the table data is displayed in the main data area of the Verification Point Editor and the Verification Point Data wizard page. The table displays the same information as the table in your application. You can edit which cells in the table are tested. Table cells in the comparison regions are shown with a gray background. You can also modify the metadata features of the table such as the column headers and row headers.

To change the data table, perform one or more of these tasks:

- To edit a value in a cell, double-click the cell and change the value.
- To modify the region of comparison of the table for testing:

1. Click **Column**, **Row**, or **Cell Selection** from the list on the toolbar above the data region to make cell selections in the table. For example, if you click **Row Selection**, and click a cell in the second row, the entire second row is selected. If you click **Cell Selection**, only that cell is selected.
2. After you select the data to compare, click the **Update Comparison Region** icon  to save the changes.

- To edit the column headers, double-click the **Value** column of the **columnHeaders** property. In the **columnHeader** dialog box, modify the required header values and close the dialog box. Similarly, you can change the row headers.
- To include the row headers or the column headers for comparison, set the **compareRowHeaders** or **compareColumnHeaders** to true.
- To change the column keys,
    1. Double-click the **Value** column of the **columnKeys** property.
    2. In the **columnKeys** dialog box, select the required key value and close the dialog box. The process of changing the row keys is similar to the preceding task.
    3.

Editing a data verification point for a tree hierarchy

When you create a data verification point and choose the **Tree Hierarchy** or **Selected Tree Hierarchy** test, the data is displayed in tree format in the main data area (right pane) of the Verification Point Editor and the Verification Point Data wizard page. The tree displays the entire tree hierarchy of your application. Each individual item is shown under its top-level item. Use the plus and minus signs to open and close the list for each top-level item. By default, all tree items appear with a check mark, which means they are tested. Checked items are tested each time you play back a script with this verification point.

- To edit the list of tree items that are tested during playback, use the check box beside each item. Checked items are tested each time you play back a script that includes this verification point. Click the **Check All** or **Uncheck All** to select or clear all the items in the list.
- To edit an item in the hierarchy, double-click the item in the tree. Edit the text in the dialog box, and then close the dialog box.

Selecting objects and data value options for data verification points

This table lists the object selection methods and the available data value options for each data type.

Table 12.

Data type	Data controls	Object selection method	Data value options and description
List	list	Select any item in the list to test the entire list	• **List Elements**: To test the contents of the entire list • **Selected List Elements**: To test only the selected items
Menu Hierarchy	• menu • menu bar	• To test a menu item and its sub-items, click the object finder on an individual top-level menu in the menu bar. • To test the entire menu hierarchy, use the object finder to select all the top-level menus.	• **Menu Hierarchy**: To test all the menus in the application and the basic properties of each menu. • **Menu Hierarchy with Properties**: To test the hierarchy and the detailed properties of each menu.
State	• check-box • toggle-button	Select the field or area in your application that contains the check box or toggle button to test.	• **CheckBox Button State** • **Toggle Button State**
Table	table	• To test the entire table, use the object finder to select any cell in the table. • To test a single cell or selected cells, select the cell or cells first, then click the Object Finder anywhere within the selected cell or cells.	• **Table Content**: To test the contents of the entire table. • **Selected Table Cells**: To test only the cells that are selected.

Table 12. (continued)

Data type	Data controls	Object selection method	Data value options and description
DataGrid	DataGrid	• To test the entire DataGrid table, use the object finder to select any cell in the table. • To test a single cell or selected cells, select the cell or cells first, then click the Object Finder anywhere within the selected cell or cells.	• **Table Contents**: Displays visible rows on the DataGrid • **Current Row**: Displays the row of the current cell on the DataGrid. • **Current DataTable**: Displays all the records without any filter. This option is displayed when the datasource is a DataSet or a DataViewManager. • **All Data**: Displays all the records without any filter. This option is displayed when the data source is a DataView.
DataGrid-View	DataGrid-View	• To test the entire DataGrid-View table, use the object finder to select any cell in the table. • To test a single cell or selected cells, select the cell or cells first, and then click the Object Finder anywhere within the selected cell(s).	• **Selected Rows -View**: Displays the content of the selected rows that are visible on the grid. • **Current Row - View**: Displays the current row contents of the DataGridView. • **All Data - View**: Displays all the visible row contents of the DataGridView • **Selected Rows - Source**: Displays the row contents of the datasource , which is mapped to the row elements of Selected-Rows collection of the DataGridView . • **Current Row - Source**: Displays the datasource row content, which is mapped to the DataGridView's current row. • **All Data - Source**: Displays all the rows in the datasource.

Table 12. (continued)

Data type	Data controls	Object selection method	Data value options and description
ToolStrip	ToolStrip	• To test a ToolStrip item and its sub-items, click the object finder on an individual top-level ToolStrip item in the ToolStrip bar.	• **Item Hierarchy**: Displays the hierarchy of the selected ToolStrip item. • **Text**: Displays text associated with the selected ToolStrip item. • **TooltipText** : Displays tooltip text that is associated with the selected ToolStrip item.
Text	text	Select the object, field or area in the application that contains the text.	• **Visible Text**: To test a text area. • **Object Visible Text**: To test the textual data on an object
Tree Hierarchy	tree	Click the object finder on any item in the tree. This verification point can test either the entire tree hierarchy or the hierarchy starting from the top of the tree through the selected item.	• **Tree Hierarchy**: To test the entire tree hierarchy • **Selected Tree Hierarchy**: To test the hierarchy starting at the top of the tree, down to the selected item.

.

Creating an image verification point

You can use an image verification point to test images in your application. When you record the verification point, a baseline image file is created. Every time you play back the script, the image is compared to see whether any changes have occurred, either intentionally or unintentionally.

About this task

Starting from 9.1.1, enhancements have been made to image verification points. Prior to 9.1.1, image verification was strictly a pixel-to-pixel comparison. When a script was recorded on one computer and played back on another computer, the image verification point sometimes failed because of system-level differences, such as screen resolution or differences in the operating system. Now, if a pixel-to-pixel comparison fails, a new image-based algorithm is applied that is more tolerable to minor changes to the image.

The algorithm returns an integer from 0 to 100 that reflects how much the two images correlate. By default, anything above 80 (or 80% correlation) is considered a pass. You can customize this percentage by adding a *"rational.test.ft.image.correlationaverage"* flag in the `ivory.properties` file, for example:

```
rational.test.ft.image.correlationaverage=70
```

In this example, anything above 70% correlation is considered a pass.

Starting from 9.1.1.1, Rational® Functional TesterHCL OneTest™ UI includes support for using Optical Character Recognition (OCR) with image verification points. This allows you to capture text along with the image.

1. Click the Record a Functional Test Script button on the product toolbar.
2. In the Recording Monitor, click Start Application to start your test application.
   See related topics about starting the test application.
3. In the application under test, locate the image that you want to test.
4. In the Recording Monitor, click Insert Verification Point or Action Command on the toolbar.
5. On the Select an Object page of the Verification Point and Action wizard, use the **Object Finder** tool to select the object. Alternatively, if the image is not based on an object that is supported by the functional test application, use the **Capture Screen Image** tool. This tool captures the full image of the screen.
   See related topics about Select an object page.
6. On the Select an Action page, click **Perform Image Verification Point** and click **Next**.
   This page is not displayed if you use the Capture Screen Image tool.
7. On the Insert Image Verification Point Command page, perform the following steps:

   a. Accept the default **Verification Point Name** or type a new name.

   b. Create one of the following types of verification points:
      - **Full image**: Select this type if the selected object or the full screen was captured using the Object Finder tool or the Capture Screen Image tool.
      - **Region of the image**: Select this type to capture a region of the image or the object using the Select Region tool. The **x** and **y** coordinates and the total width and height of the selected region are captured as the image verification point.
      - **Text on the image**: Select this type to use OCR to capture the text on the image. Click **Select Region**, and follow the guidance in the tool for instructions on how to select the region.

   c. Click **Next**.
      The Verification Point Data page displays the captured image.

8. Click **Finish**.
   The verification point is recorded and added to the script.

   Note:

   In the unified report, an image verification point name is displayed in the following format:

   ```
   Verify Image <window_name> <verification_point_type> <property_name>
   ```

9. After you record any other verification points or actions, stop your recording by clicking the Stop Recording button on the Recording Monitor toolbar.

Using OCR to test application text

With the optical character recognition (OCR) support in 9.1.1.1, you can use the image verification point to test text in your application.

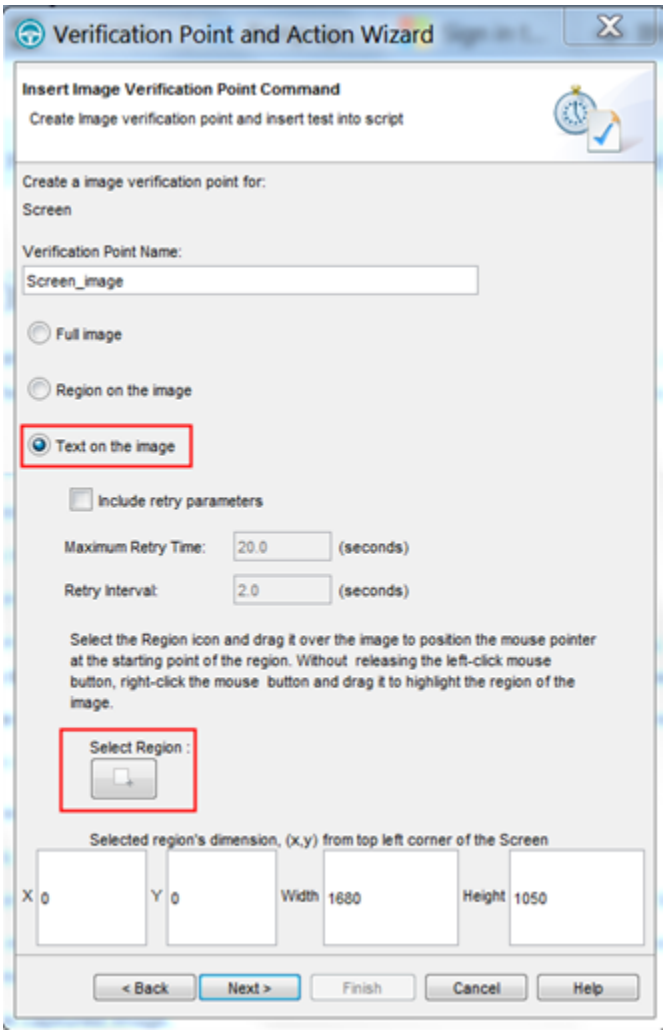
Before you begin

For OCR support, you must install Microsoft™ Visual C++ 2015 Redistributable from the Microsoft™ Download Center.

About this task

- For 9.1.1.1, OCR support is available on Windows™ for testers using the Eclipse Integrated Development Environment (IDE). OCR support is not yet available for testers using the Visual Studio IDE or Linux™ operating systems.
- OCR support is not currently available for languages other than English.

1. Follow Step 1 on page 915 to Step 6 on page 915 in the topic on Creating an image verification point on page 914.
2. For the Insert Image Verification Point Command in Step 7 on page 915, choose **Text on the image** and click the **Select Region** icon. Follow the guidance in the tool for instructions on how to select the region.



For best results, follow these guidelines:

- Capture a region slightly larger than the image itself.
- Repeat the image capture as necessary until you get satisfactory results.
- Some fonts work better than others, so if necessary, try changing the font.

3. Complete the remaining steps in the topic on Creating an image verification point on page 914.

Verification Point Editor

The Verification Point Editor lets you view and edit verification point data. You can open the Editor by double-clicking a verification point in the Script Explorer window. The Editor banner displays the name of your verification point.

You can specify color settings for several elements in the Verification Point Editor.

The following sections explain the parts of the Verification Point Editor window, and the toolbars.

Metadata

The metadata is displayed in the left pane of the window. It displays a set of properties that define how specific data is managed. This grid can be edited. For example, you can edit the 'ignore case' or 'white space rule' in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

Main toolbar

The toolbar at the top of the Verification Point Editor has five buttons.

 **File: Save** -- Saves any edits you have made.

 **File: Revert** -- Reverts to the state of the data at the last save you made. If you have not saved edits since opening the verification point, it will revert to the state it was in when opened. If you have done editing and made saves, it will revert to the state at your last Save.

 **Hide/Show TestObject Info** -- Toggles the display of the **Test Objects** and **Recognition Data** panes of the Editor window. When this information is hidden, the entire Editor window is used for the main data area. This is a sticky setting--the next time you open the Editor it will appear as you last set it. However, note that if your Test Objects tree has multiple nodes, the Verification Point Editor will show these panes again the next time you open it, regardless of this setting.

 **Replace Baseline** -- Replaces the baseline image with a new image. The new image will become the baseline for future playbacks. The Verification Point and Action Wizard is invoked for recapturing the image verification point.

 **Help** -- Brings up the Help for the Verification Point Editor. You can open the Rational® Functional TesterHCL OneTest™ UI help at any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Menu bar

The menu bar contains the same commands that are represented with the toolbar buttons described in this topic.

File -- These are the same **Save**, **Revert**, **Check Out**, and **Exit** commands as the buttons listed above in the Main Toolbar section.

Edit -- These are the same commands as the buttons listed below in the Properties Verification Point section.

Test Object > Highlight -- If your test application is open, you can select an object in the Test Objects tree and then click this command to see the object highlighted in the application. Use this feature if you need to verify an object in the application.

Preferences > Toolbars -- Toolbars controls the display of the toolbars. Hides/Displays the File, Metadata and Help toolbars. **Test Object Appearance on the Tree** displays the Edit Test Object Description dialog box on page 1757, which enables you to customize the text displayed for each object in the Test Object Hierarchy. **Hide TestObject Info** toggles the display of the **Test Objects** and **Recognition Data** panes of the Editor window.

Help -- Displays the Help for the Verification Point Editor. You can open the Rational® Functional TesterHCL OneTest™ UI Help any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Main data area

The right pane of the Verification Point Editor is where the verification point data is displayed. For example, in the case of a Properties verification point, the **Property** and **Value** columns are displayed here. This is where you edit the verification point data.

There are seven different types of displays you can get from recording verification points, as described in the following sections.

Properties Verification Point -- Grid Display

When you create a Properties verification point, the object properties are displayed in a grid format. See Creating a Properties Verification Point on page 904 for information on recording it. The properties that are shown in the grid belong to the object that is highlighted in the **Test Objects** tree. The properties appear in the left column and their values appear in the right column. You can edit which properties get tested in the **Property** column by checking a check box for a property, and can edit the property values themselves in the **Value** column.

By default, all properties will appear with no checkmark, which means they will not be tested. Choose which properties you want to test by checking each of them. Checked properties will be tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button  above the grid. Use the **Uncheck All** button  above the grid to clear all properties. Depending on how many properties you want to test, it is often easiest to either select or clear all of them using one of those buttons, and then individually select or clear exceptions. It's a good idea to just test the specific properties you are interested in when you use a Properties verification point.

The grid uses a nested tree hierarchy. If a folder shows up on the list, you can expand it by double-clicking on it or selecting the expand icon. If you select or clear the folder icon itself, all the properties underneath it will be tested or not tested.

To edit a value, double-click the grid cell. That cell will then be editable. Click outside the cell to make the edit take effect. In most cases double-clicking a value makes the cell an editable field, and you can just change the value. In some special cases, another dialog box comes up containing the information. For example, if the property is color, when you double-click the color value, the standard Color dialog box opens. Make your edit there and close the Color box. In other cases, a drop-down list may appear in the **Value** column when you double-click a value. For example, values that are either true or false will appear in a drop-down list. If the value is a string or a complex value type, you can right-click the value and select **Open** to display the value in a separate window, which enables you to see long lines of text and makes it easier to edit.

Note: You can change a property value to a regular expression or numeric range using the Verification Point Editor. For information, see Replacing an Exact-Match Property with a Pattern.

The grid has the following toolbar buttons for the Properties verification point display. These buttons act only on the currently displayed data.

Cut -- Cuts the selected property. It is placed on the Editor clipboard and can be pasted.

Copy -- Copies the selected property to the Editor clipboard.

Paste -- Pastes the cut or copied property. It will be inserted into the display in alphabetical order.

Delete -- Deletes the selected property. It will not be retained on the clipboard.

Case Sensitive Regular Expression -- Toggles case-sensitive comparison on and off.

Convert Value to Regular Expression -- Converts the recognition property value in the **Updated Test Object Properties** grid to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

Convert Value to Numeric Range -- Converts the recognition property value in the **Updated Test Object Properties** grid to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

Evaluate Regular Expression -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.

Convert Value to dataset Reference/ **Undo dataset Reference** -- Uses a dataset reference to use a dataset instead of a literal value in a verification point. Cancels the dataset reference in the verification point. See About dataset References and Verification Points on page 951.

Check All -- Puts a checkmark in front of every property in the list. Checked properties will be tested each time you play back the script with this verification point.

Uncheck All -- Clears the checkmark in front of every property in the list. Cleared properties will not be tested when you play back the script with this verification point.

Hide the Unchecked Properties/Show All Properties -- Click **Hide the Unchecked Properties** to hide the cleared properties. Then you will only see the properties that will be tested. Click **Show All Properties** to display all properties, including any cleared ones.

The grid has the following pop-up menu commands for the Properties verification point display. To access them, right-click a value in the **Value** column.

Open -- If the value is a string or a complex value type, this will display the value in a separate window, which enables you to see long lines of text and makes it easier to edit.

Case Sensitive Regular Expression -- Toggles case-sensitive regular expression comparison on and off.

Evaluate Regular Expression -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.

Convert Value to Regular Expression -- Converts the property value to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

Redo/Undo Regular Expression -- Redoes or cancels the regular expression conversion.

Convert Value to Numeric Range -- Converts the property value to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

Undo Numeric Range -- Redoes or cancels the numeric range.

Convert Value to dataset Reference - - Uses a dataset reference on page 951 to use a dataset instead of a literal value in a verification point.

Undo dataset Reference -- Cancels the dataset reference in the verification point on page 951.

Data Verification Point--Menu Hierarchy Display

When you create a Data verification point and choose the Menu Hierarchy or Menu Hierarchy with Properties test, the menus are displayed in a tree format in the main data area (right pane). Menu Hierarchy and Menu Hierarchy with Properties are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be shown.

The tree will display the entire menu hierarchy of your application, or one top-level menu and its sub-items, depending on how you recorded the verification point. If you chose the whole menu bar, each top-level menu will be shown from top to bottom in the tree in the order they appear from left to right in the menu bar. Each individual menu item is shown under its top-level menu. Use the plus and minus signs to open and close the list for each top-level menu.

By default, all menu items will appear with a check mark, which means they will be tested. Checked items will be tested each time you play back a script with this verification point, and cleared items will not be tested. You can check all menu items by clicking the **Check All** toolbar button above the tree. Use the **Uncheck All** button to clear all items.

The **Cut**, **Copy**, **Paste**, **Delete** , **Check All**, and **Uncheck All** toolbar buttons above the tree apply to the selected menu item in the tree hierarchy, and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

Data Verification Point--Text Display

When you create a Data verification point and choose the Visible Text test, the text is displayed in a text box format in the main data area (right pane). Visible Text is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

The text is displayed in a text box that can be used like a very basic text editor. You can type and edit directly in this text box. To edit the verification point data, make your edits to the text in this area.

Data Verification Point--Table Display

When you create a Data verification point and choose the Table Contents or Selected Table Cells test, the table data is displayed in a table in the main data area (right pane). Table Contents and Selected Table Cells are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these may be shown.

The table displays the same information as the table in your application. To edit the verification point data, double-click any cell in the table to edit that cell.

The **Cut**, **Copy**, **Paste**, and **Delete** toolbar buttons above the table area apply to the selected row(s), and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

You can right-click a table item to access a pop-up menu. The commands are the same as those listed above in the **Properties Verification Point--Grid Display** section.

Data Verification Point--Tree Hierarchy Display

When you do a Data verification point and choose the Tree Hierarchy or Selected Tree Hierarchy test, the data is displayed in a tree format in the main data area (right pane). Tree Hierarchy and Selected Tree Hierarchy are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be shown.

The **Cut**, **Copy**, **Paste**, **Delete** , **Check All**, and **Uncheck All** toolbar buttons above the tree apply to the selected item in the tree hierarchy, and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

Data Verification Point--List Display

When you create a Data verification point and choose the List Elements test, the data is displayed in a list format in the main data area (right pane). List Elements is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

The toolbar buttons above the list are the same ones that are found in the object properties grid described above in the Properties Verification Point--Grid Display section. The buttons work the same as described there, except they apply to the selected list item(s). The **Cut**, **Copy**, **Paste**, and **Delete**, **Check All**, and **Uncheck All** toolbar buttons are only applicable within the Verification Point Editor. (It does not use the system clipboard.) The **Insert** button is described above.

Data Verification Point--State Display

When you create a Data verification point and choose the CheckBox Button State or Toggle Button State test, the data is displayed in a list format in the main data area (right pane). CheckBox Button State or Toggle Button State are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

Test object data in the Verification point editor window

While inserting the verification points, if you have not checked the **Record Test Object relative Verification Points** option available in the General Recorder page on page 843 of the **Windows > Preferences** window, you can view the following test object data in the Verification Point editor:

- Test objects
- Recognition and Administrative data

Test objects

This is the upper left pane of the Verification Point Editor window. It's a partial version of the script's object map. This hierarchical display includes only the objects in your verification point. You cannot edit the Test Objects tree. For a Properties verification point, you can choose an object within it and edit its properties in the properties list in the right pane.

You can double-click folders in the tree to expand and collapse the objects beneath them. Click an individual object in the tree to see its properties in the properties list.

The check boxes to the left of each node indicate whether that node will be tested or not. Checked items get tested.

Note: If your test application is open, you can select an object in the Test Objects tree and then click **Test Object > Highlight** or right-click an object and click **Highlight** from the Verification Point Editor menu to see the object highlighted in the application. Use this feature if you need to verify an object in the application.

Recognition and Administrative data

This is the lower left pane of the Editor window. The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI and is not editable. The **Administrative** tab displays internal administrative data of the object and is not editable. These properties are used to manage and describe the test object. Recognition and administrative data are the properties from the script's object map used to locate and manage this test object in the context of the associated script. You can use this information to figure out what test object this is in the associated application under test.

The **MetaData** tab displays a set of properties that define how specific data is managed. This grid can be edited. For example, you could edit the 'ignore case' or 'white space rule' in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

The Recognition and Administrative properties are a snapshot of the object map properties for the test object at the time the verification point was created. They become historical information as the application evolves.

Verification point comparator

You can use comparators to compare verification point data after you play back a script with a verification point and to update the baseline file. If the verification point fails, the comparator shows both expected and actual values so

that you can analyze the differences. You can then load the baseline file and edit or update it with the values from the actual file.

Opening the verification point comparator

Click **View Results** in the Rational® Functional TesterHCL OneTest™ UI HTML log to open the comparator.

Notes:

- If you encounter an error regarding the Java™ plugin when trying to launch the comparator from **View Results** in the HTML log, you must configure your plugin properly.
- You must use Microsoft Internet Explorer to open **View Results** as browsers such as Mozilla Firefox and Google Chrome are not supported.
- With automatically enabled test environments, you cannot open the verification point comparator by clicking **View Results** in the functional test HTML log. In such cases, open the corresponding project log file from the functional test project log in the Functional Test Projects view.

Using verification point comparator for functional test scripts played back from Rational® Quality Manager

If you play back the script from Rational® Quality Manager, and click **View Results** in the detailed log to open the comparator, you are prompted to log in to Rational® Quality Manager. This occurs when you use the **Load baseline to edit** or the **Replace baseline with actual value** functions.

Note: To open the comparator from the Rational® Quality Manager detailed playback log, ensure that Rational® Functional TesterHCL OneTest™ UI is installed on the workstation where you are opening the log. Additionally, ensure that the Next-Gen plugin is disabled on the workstation.

When you open the comparator, the **Log In to Rational Quality Manager** dialog box is displayed and the Rational® Quality Manager server name and project area are displayed. You must specify a valid user name and password to log in to Rational® Quality Manager.

The **Log In to Rational Quality Manager** dialog box is displayed only the first time you use **Load baseline to edit** or the **Replace baseline with actual value** functions during an active Windows session. You are not prompted to log in a second time unless you have started a new Windows session and logged on to Windows as a different user.

Comparing verification points after playback

If you have one failed verification point, and you are using a log, select the log in the Functional Test Projects view. Right-click the log, and click **Failed Verification Points**. If you have one failed verification point, and you are using a log, select the log in the Solution Explorer. Right-click the log, and click **Failed Verification Points**. The verification point comparator is displayed.

If you have multiple failed verification points, and you are using a log, the **Results for Verification Points** wizard is displayed. Click a failed verification point in the list and click **View Results** or **Finish**. The comparator banner displays the name of your verification point.

You can specify color settings for several elements in the verification point comparator.

To edit verification point data, you must load the baseline by clicking the **Load Baseline to Edit** toolbar button  .

The verification point comparator window

The following sections describe the various components of the verification point comparator window and the toolbars.

Metadata

Metadata is displayed in the left pane of the verification point comparator window. It displays a set of properties that define how specific data is managed. You can edit this grid. For example, you could edit the *"ignore case"* or *"white space rule"* in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

Main toolbar

The toolbar at the top of the verification point comparator has six buttons.

 **File: Save**: Saves any changes you have made.

 **File: Revert**: Reverts to the state of the data at the last save you made. If you have not saved any changes since opening the comparator, it reverts to the state when it was opened. If you have edited and saved the changes, it reverts to the state at your last Save.

 **Load Baseline to Edit**: Loads the baseline file so you can edit it. The baseline values are displayed instead of the expected values. These values can be edited individually or replaced with actual values.

 **Replace Baseline with actual value**: Replaces the baseline values with all the values in the actual file. Then those values will become the baseline for future playbacks. If you want to replace only some of the values, edit them individually. This command replaces the entire file.

 **Hide/Show TestObject Info**: Toggles the display of the **Test Objects** and **Recognition Data** panes of the comparator window. When this information is hidden, the entire comparator window is used for the main data area. This is a sticky setting and is displayed as you last set it when you open the comparator later.

Note: If your test objects tree has multiple nodes, the verification point comparator displays these panes again the next time you open it, regardless of the **Hide/Show TestObject Info** setting.

Help: Displays the help documentation for the verification point comparator. You can open the Rational® Functional TesterHCL OneTest™ UI help any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Menu bar

The menu bar contains the same commands that are represented with the toolbar buttons described in this topic.

File: Displays the commands **Save**, **Revert**, **Baseline**, and **Replace**.

Edit: Displays the commands **Check All**, **Uncheck All**, and **Hide**. This menu is grayed out until you load the baseline for editing (using the **Load Baseline to edit** toolbar button).

Difference: Displays the commands **First**, **Previous**, **Next**, and **Last**.

Test Object > Highlight: You can use this menu item if you need to verify an object in the application. If your test application is open, you can select an object in the test objects tree and then click this command to see the object highlighted in the application.

Preferences : Toolbars: You can use this menu item to control the display of the toolbars.

- **Test Object Appearance on the Tree**: Displays the **Edit Test Object Description** dialog box, which allows you to customize the text displayed for each object in the Test Object Hierarchy.
- **Hide TestObject Info**: You can use this command to toggle the display of the **Test Objects** and **Recognition Data** panes of the comparator window.

Help: Displays help documentation for verification point comparator.

Main data area

The right pane of the verification point comparator displays the verification point data. For example, in the case of a properties verification point, the **Property** and **Value** columns are displayed here. You can compare the verification point data here. If the verification point fails when you play back the script, the expected and actual values are displayed, irrespective of the type data display being used. In certain cases, the expected values are shown on the left pane and the actual values are shown on the right pane of the verification comparator window. In other cases, the values are displayed contiguously (such as nodes in a tree view), and the expected and actual values are shown in different colors if they are different. The expected value is red and the actual value is green in color. The actual values those that were recorded when you playback the script.

You can get seven types of displays from recording verification points, as described in the following sections, after the next section, Navigation Toolbar Buttons.

Navigation toolbar buttons

These four navigation buttons jump to the differences between the expected and actual files or the baseline and actual files. Differences are shown in red. The currently selected difference is highlighted.

 **Jump to First Difference**: Goes to the first difference in the expected/baseline and actual files.

 **Backward to Previous Difference**: Goes backward to the previous difference in the expected/baseline and actual files.

 **Forward to Next Difference**: Goes forward to the next difference in the expected/baseline and actual files.

 **Jump to Last Difference**: Goes to the last difference in the expected/baseline and actual files.

You can get the following types of displays after recording a verification point.

Properties verification point : grid display

When you create a properties verification point, the object properties are displayed in a grid format. The properties displayed on the grid belong to the object that is highlighted in the **Test Objects** tree. The properties appear are displayed in the left column and their values appear are displayed in the right column of the object properties grid.. You can edit which properties get tested in the **Property** column, and can edit the property values themselves in the **Value** column.

Properties with no check mark are not tested. You can select which properties you want to test by checking each of them. The checked properties are tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button  . You can use the **Uncheck All** button  to clear all properties. Depending on how many properties you want to test, it is often easiest to either select or clear all of them using one of those buttons, and then individually select or clear exceptions.

The grid uses a nested tree hierarchy. If a folder shows up on the list, you can expand it by double-clicking it or selecting the expand icon. If you check or clear the folder icon itself, all the properties underneath are either tested or not tested.

To edit a value, you must double-click the grid cell. Click outside the cell to make the edit take effect. In most cases, double-clicking a value makes the cell an editable field, and you can just change the value. In some special cases, another dialog box is displayed which contains the information. For example, if the property is color, when you double-click the color value, the standard color dialog box is displayed. You must your edit there and close the color box. In other cases, a drop-down list might be displayed in the **Value** column when you double-click a value. For example, values that are either true or false are displayed in a drop-down list.

The grid has the following toolbar buttons for the properties verification point display. In the comparator, these buttons are displayed only when you are editing the baseline.

 **Check All**: Includes a check mark in front of every property in the list. Checked properties are tested each time you play back the script with this verification point. Only checked properties are compared in the Comparator.

Uncheck All: Clears the check mark in front of every property in the list. Do not test the cleared properties when you play back the script with this verification point.

Hide the Unchecked Properties/Show All Properties: Click **Hide the Unchecked Properties** to hide the cleared properties. Then you only view the properties that are tested. Click **Show All Properties** to display all properties, including any cleared ones.

The grid has the following pop-up menu commands for the properties verification point display. To access them, right-click a value in the **Value** column.

Open: Displays the value in a separate window if the value is a string or a complex value type which enables you to see long lines of text and makes it easier to edit.

Case Sensitive Regular Expression: Toggles case-sensitive regular expression comparison on and off.

Evaluate Regular Expression: Displays the regular expression evaluator, which enables you to test the regular expression before you use it in a verification point.

Convert Value to Regular Expression: Converts the property value to a regular expression.

Undo/Redo Regular Expression: Cancels or redoes the regular expression conversion.

Convert Value to Numeric Range: Converts the property value to a numeric range.

Undo Numeric Range: Cancels the numeric range.

Convert Value to dataset Reference: Uses a dataset reference to use a dataset instead of a literal value in a verification point.

Undo dataset Reference: Cancels the dataset reference in the verification point.

Replace Baseline On Current Selection: Replaces the baseline value with the actual value for just the selected property. This is a per-property version of the **Replace Baseline With Actual Value** toolbar button .

Compare object properties

To compare object properties, look at the expected or baseline values and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which are displayed in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : menu hierarchy display

When you create a data verification point and choose the menu hierarchy or menu hierarchy with the properties test, the menus are displayed in a tree format in the main data area. Menu hierarchy and menu hierarchy with properties

are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be displayed.

The tree displays the entire menu hierarchy of your application, or one top-level menu and its subitems, depending on how you recorded the verification point. If you chose the whole menu bar, each top-level menu is displayed in the tree, in the same order they are displayed in the menu bar. Each individual menu item is displayed under its top-level menu. You can use the plus and minus signs to open and close the list for each top level menu.

To edit a menu, double-click it in the tree. You must load the baseline first before doing this. The menu properties displayed in a grid, which you can then edit. You can edit the actual values by double-clicking a value in the **Value** column. You can also edit the list of properties that are tested during playback by using the check box beside each property. The checked items are tested. The toolbar buttons above the grid are the same ones that are found in the object properties grid, except for **Hide/Show**. The buttons work the same, except they apply to the selected menu property or value.

Compare menu hierarchy data

To compare menu hierarchy data, look at any differences shown in red and green. The expected values are displayed in red, and the actual values are shown underneath them in green. The actual values are what were captured when you played back the script. If the descriptions for the expected and baseline values are the same, but if there are some differences in their properties, the node is displayed as blue in color. You can use the navigation buttons to navigate to all the differences. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : text display

When you create a data verification point and choose the Visible Text test, the text is displayed in a text box format in the main data area. For example, visible text. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be displayed.

The text is displayed in a text box area. You cannot edit directly in this area. To edit the verification point data, click **Edit Text**  above the data display. You must load the baseline file before doing this. A small text editor containing the text is displayed. You can edit the text in this editor, and when you close it, the edited text is displayed in the baseline column of the comparator.

Compare text data

To compare text data, look at the expected and actual values columns. The actual values are those that were captured when you play back the script. You can use the navigation buttons to navigate to all the differences, which is displayed in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : table display

When you create a data verification point and choose the table contents or selected table cells test, the table data is displayed in a table in the main data area. Table Contents and selected table cells are two examples. The list of tests is displayed in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these may also appear.

The table displays the same information as the table in your application. To edit the verification point data, double-click any cell in the table to edit that cell. You must load the baseline file before doing this.

You can also edit which cells in the table get tested. Table cells that are within the comparison regions are shown with a grey background. If you are testing the entire table, all cells will be grey. You can use the drop-down list in the toolbar above the data region as a selection mechanism. (This doesn't show up until you load the baseline.) Choose **Column**, **Row**, or **Cell Selection** in the list, then make your selections in the table. For example, if you select **Row Selection**, when you click a cell in the second row, the whole second row will be selected. If you had chosen **Cell Selection**, only that cell would have been selected. After you select the data you want to compare, click the **Update Comparison Region** button  to have your changes take effect.

The **Cut**, **Copy**, **Paste**, and **Delete** toolbar buttons above the table area apply to the selected row(s), and are only applicable within the Verification Point Comparator. (It does not use the system clipboard.)

You can right-click a table item to access a pop-up menu. The commands are the same as those listed above in the **Properties Verification Point--Grid Display** section.

There are features in the **Metadata** tab that you can also use to edit the table data. For example, you can edit the table's column headers or row headers by accessing them in the **MetaData** tab. To edit column headers, double-click the **Value** column of the **columnHeaders** property. A small editor opens that lets you edit the headers. The row headers work the same way if your table has them. Double-click the **rowHeaders Value** to edit them. In order for the column headers to be compared, you must change the **compareColumnHeaders** property to **true** in the MetaData tab. The **compareRowHeaders** value works the same way to indicate whether row headers will be compared.

If you double-click the **Value** of the **compareRegions** property in the Metadata tab, an editor will open showing the selected regions of your table. For selected sells, it shows the row index or key value pairs and the column header or index of each selected cell. For selected rows, it shows the row index or key value pairs. For selected columns, it shows the column header or index. Using this compare regions editor is another way you can select which regions get compared. If you click the **Compare All Cells** button  in this editor, all of the table cells will be tested.

If your table supports row keys or column keys, you can edit those and insert keys by double-clicking on the **columnKeys** and **rowKeys** values in the **Metadata** tab.

Compare table data

To compare table data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which appear in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point -- tree hierarchy display

When you create a Data verification point and choose the Tree Hierarchy test, the data is displayed in a tree format in the main data area. For example, Tree Hierarchy. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this might be displayed.

The tree displays the entire tree hierarchy in your application or the part of the tree selected when you create the verification point. Each item in the tree is displayed in the same order it is displayed in your application. Each individual item is displayed under its top-level item. You can use the plus and minus signs to open and close the list for each top-level item.

To edit an item in the hierarchy, double-click it in the tree. A small text box is displayed, which you can use to edit the item.

Compare tree hierarchy data

To compare tree hierarchy data, look at any differences displayed in red and green. The expected values are displayed in red, and the actual values are displayed in green. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences.

Data verification point : list display

When you create a data verification point and choose the List Elements test, the data is displayed in a list format in the main data area. List Elements is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this might be also displayed.

The list displays the same information as the list in your application, and in the same order. To edit a list item, double-click it in the list display. (If you have not done so, you must load the baseline first.) You can also edit the list of which items get tested during playback by using the check box beside each item. The checked items are tested.

The toolbar buttons preceding the list are the same ones that are found in the object properties grid described above in the Properties Verification Point : Grid Display section. The buttons work the same as described there, except they apply to the selected list item(s).

You can right-click a table item to access a pop-up menu. The commands are the same as those listed preceding the **Properties Verification Point : Grid Display** section.

Compare list data

To compare list data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which are shown in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : state display

When you create a data verification point and choose the Check Box Button State or Toggle Button State test, the data is displayed in a list format in the main data area. Check Box Button State or Toggle Button State are two examples. The list of tests displayed in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this may be also displayed.

Compare state data

To compare state data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can edit the baseline values or replace the baseline with the actual file.

Test object data in the Verification point comparator window

While inserting the verification points, if you have not checked the **Record Test Object relative Verification Points** option available in the General Recorder page of the **Windows > Preferences** window, you can view the following test object data in the Verification Point comparator:

- Test objects
- Recognition and Administrative data

Test objects

This is the upper left pane of the Verification Point Comparator window. It is a partial version of the script's object map. This hierarchical display includes only the objects in your verification point. You cannot edit the Test Objects tree. You can choose an object within it and edit its properties or data in the right pane of the Verification Point Comparator window.

You can double-click folders in the tree to expand and collapse the objects beneath them. You must an individual object in the tree to see its properties or data in the right pane.

The check boxes to the left of each node Verification Point Comparator window indicate whether that node is tested or not. Checked items get tested. After you load the baseline to edit, you can select or clear items.

Note: If your test application is open, you can select an object in the Test Objects tree and then click **Test Object > Highlight** from the Verification Point Comparator menu to see the object highlighted in the application. You must use this feature if you need to verify an object in the application.

Recognition and Administrative data

This is the lower left pane of the Verification Comparator window. The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI and is not editable. Some of these properties are the recognition properties that were listed in the **Select an Object** tab of the Verification Point and Action Wizard when you created the verification point. The **Administrative** tab displays internal administrative data of the object and is not editable. These properties are used to manage and describe the test object. Recognition and administrative data are the properties from the script's object map used to locate and manage this test object in the context of the associated script. You can use this information to determine what test object this is in the associated application under test.

The **MetaData** tab displays a set of properties that define how specific data is managed. This grid can be edited if you load the baseline. For example, you could edit the *"ignore case"* or *"white space rule"* in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

The Recognition and Administrative properties are a snapshot of the object map properties for the test object at the time the verification point was created. They become historical information as the application evolves.

Related reference

Edit Test Object Appearance dialog box on page 1757

Related information

Comparing and updating verification point data using the Comparator on page 933

Enabling the Java plug-in of a browser on page 800

Viewing logs in the Projects view on page 1501

Editing test object descriptions

Replacing an exact-match property with a pattern

Comparing and updating verification point data using the Comparator

Use the Verification Point Comparator to compare verification point data after you play back a script with a verification point and to update the baseline file. If the verification point failed, the Comparator shows both the expected and actual values, so you can analyze the differences. You can then load the baseline file and update it with the values from the actual file.

About this task

To open the Comparator, click the **View Results** link in the Rational® Functional TesterHCL OneTest™ UI HTML log. For information, see the related topic about viewing results in the log. The Comparator banner displays the name of your verification point.

1. Play back the script that contains the verification point on a new build of the application under test.
   Result
   The log for the playback is displayed. For information about setting the option that makes the log open automatically after playback, see Logging Preferences Page.
2. Open the log for the verification point.
   - If you are using the HTML log, click the **View Results** link. (Note that if you experience an error regarding the Java™ plug-in when trying to start the Comparator from the **View Results** link in the HTML log, verify that your plug-in is configured properly.
3. Open the Comparator from the log.
   - If one verification point failed, select the log in the Projects viewSolution Explorer, right-click the log, and click **Failed Verification Points**.
   - If you have more than one failed verification points the Results for Verification Points wizard opens. Click a failed verification point in the list and click **View Results** or **Finish**.

   Result
   The Verification Point Comparator opens to display that verification point. The Comparator includes the expected and actual data values. The expected values were tested. The actual values were captured in the application during playback. If the verification point failed, the differences are shown in red.

For the data verification point types list, table and text, the expected values are displayed on the left and the actual values are displayed on the right. The differences are shown in red. For the data (menu hierarchy) and data (tree hierarchy) verification points, the expected and actual values are shown contiguously. The expected values of the differences are shown in red, and the actual values of the differences are shown underneath them in green.

4. Look at the two data files to compare any differences between the expected and actual files. By analyzing differences in the Comparator, you can determine if they are intentional changes to the application or defects. To navigate through the differences, use the navigation buttons on the toolbar above the data display.
5. If you want to edit the baseline file to update the information for future playbacks, you must load the baseline file. Click **File > Baseline** or click the **Load Baseline to Edit** toolbar button  .
   The baseline file replaces the expected file on the left side of the display.
6. To edit individual items in the data, edit them in the baseline (left) column of the display. When you finish editing the data, click **Save**.

   For a Data (Text) verification point, click **Edit Text**  to start a text editor to make your edits. For other verification point types, you can edit directly in the baseline data display.
7. For a Properties verification point, if you determine that the baseline value and actual value for a specific property are different, you can update the baseline value. In the Verification Point Comparator, right-click the property where the values are different, and click **Replace Baseline on Current Selection**.
8. If you determine that all the differences reflect intentional changes to the application under test, and you want to update the baseline to reflect the changes, you can use the **Replace Baseline with Actual Value** toolbar button  to replace the entire baseline file.
9. If you have made any individual changes to the baseline data file (not by using the **Replace Baseline with Actual Value** command), click **File> Save** to save the changes.
10. When you finish comparing and updating verification point data, click **File > Exit** to exit the Verification Point Comparator.

Inserting verification points into the script using the application visuals

While working with the simplified test scripts, you can insert verification point using the application visuals.

Inserting a data verification point into a script by using an application visual

You can add data verification points into simplified scripts by using an application visual instead of opening the test application again or by using the verification point editor.

Before you begin

Prerequisite: To use application visuals for inserting data verification points into the script, the capturing of verification on data feature must be enabled in the **Preferences** page before recording the script.

1. Select the test line in the script editor that refers to the required application visual of the test application.
   The application visual that contains the control is displayed in the Application view.

Note: The thumbnail pane in the Application view displays images of all the application visuals that are captured for the project. You can select the required application visual image in the thumbnail pane and view the application visual in the Application view.

2. In the Application view, move the mouse pointer over the control on which the data verification point must be captured.
   Result
   The control is highlighted in red.
3. Right-click and select **Insert verification point > Data Verification Point**.
   The verification point wizard is displayed if you have enabled the **Show Verification Point Dialog** option in the **Preferences** page. Set the data verification point properties in the verification point wizard:
   a. On the **Verification Point Wizard** page, accept the default **Data Value** or select one of the available options from the list.
   b. Change the **Verification Point Name**, if required, and click **Next**.
   c. On the **Verification Point Data** page, edit the data if required.
   d. **Optional:** To use a dataset reference instead of a literal value for a data verification point:
      i. Right-click a property, and then click **Convert Value to dataset Reference**. **The dataset Reference Converter** dialog box opens.
      ii. Type a new name for the dataset variable or select the variable from the **dataset Variable** list for the verification point to reference in the dataset.
      iii. Select **Add value to new record in dataset** to add the value of the verification point to a new record (row) in the dataset.
      iv. Click **OK**, and close the text box.
   e. Click **Finish**.

   Result
   The verification point is inserted into the test script. You can drag the test line in the script editor to arrange the test lines in the required sequence for playback.
4. Click **File > Save** to save the modified test script.

Inserting an image verification point into a script by using an application visual

You can add image verification points to a script by using application visuals instead of opening the test application again.

Before you begin

To use the application visuals for inserting image verification points to the script, the application visuals feature must be enabled in the Preferences page before recording the script.

1. Select the test line in the script editor that refers to the required application visual of the test application. The application visual that contains the control is displayed.

   Note: The thumbnail pane in the Application view displays images of all the application visuals that are captured for the project. You can select the required application visual image in the thumbnail pane and view the application visual in the Application view.

2. In the Application view, move the mouse pointer over the control to capture for the image verification point.
   Result
   The control is highlighted in red.
3. Right-click and select **Insert verification point > Image Verification Point**.
   Result
   The image verification point is inserted into the test script. You can drag the test line in the script editor to arrange the test lines in the required sequence for playback.
4. Click **File > Save** to save the modified test script.

Creating group verification points

You can insert group verification points for all the controls that are in the application visual.

Before you begin
Prerequisites:

- The group verification point feature in the application visual is available if the **Enable capturing of verification on test data** option is enabled when the script was recorded.
- Verify if all the required control types in the application visual are the default controls for group verification point. For more information about the list of default controls and the procedure to add the control types, see the group verification points topic.

1. Select the test line in the script editor that refers to the required application visual of the test application. The application visual is displayed in the Application view.
2. Select the application visual, right-click and select **Insert Verification Point > Group Verification Point**.
   Result
   A set of test lines to capture verification points for all the controls is added to the test script.
3. Click **File > Save** to save the modified test script.
   The verification point test lines are inserted into the script for all the controls that are set as default controls for group verification points.

Group verification points

You can create verification points for all the controls of an application visual using the Group Verification Point feature.

From an application visual, you can select the controls individually and insert the data or image verification points. Using the group verification point feature in the Application view, you can insert verification points for all the controls in the application visual. By default, group verification points can be inserted for the following type of controls in the application visual:

- Text
- Button
- ToggleButton
- CheckBox
- ComboBox
- RadioButton
- Label
- Link
- ComboListBox

If you want to insert group verification points for any other type of controls such as a tree control, you must add it to the control list in the **defaultGroupVp.rftssvp** file.

To insert a control that is not listed as the default control for group verification points:

1. Close Rational® Functional TesterHCL OneTest™ UI.
2. Open the **defaultGroupVp.rftssvp** file available at *`product installation directory`*`\FunctionalTester\bin` location using the notepad editor.
3. Add the control to the file in the format `<role roleName="control name"/>`. For example: To add a tree control to the default group verification point list, add the line `<role roleName="Tree"/>`.
4. Save the file and then open Rational® Functional TesterHCL OneTest™ UI.

Note: You can refer to the control names in the **defaultVPType.rftssvp** file that is also available at `<product installation directory>\FunctionalTester\bin` location.

Driving functional tests with external data

This section describes techniques you can use to data-drive functional tests with external data.

Data-driving tests overview

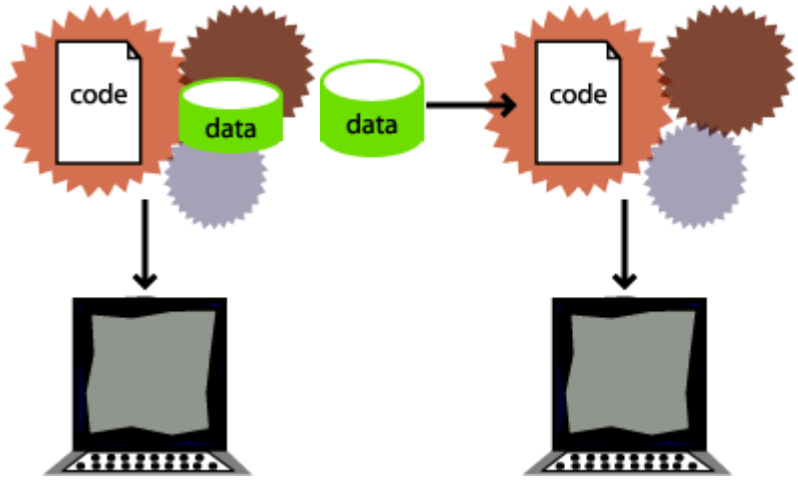
When you data-drive a test, the script uses variables for key application input fields and programs instead of literal values so that you can use external data as you data-drive the application you are testing.

Data-driven testing uses data from an external file, a dataset, as input to a test. A dataset is a collection of related data records. When you data-drive test, datasets supply data values to the variables in a test scripts during test script playback.

Because data is separated from the test script, you can:

- Modify test data without affecting the test script
- Add new test cases by modifying the data, not the test script
- Share the test data with many test scripts

The diagram on the left shows a test script, which uses data with hard-coded, literal references in the test script. The diagram on the right shows a data-driven test script that uses data from an external file, a dataset.



Hard-coded test script with literal references Data-driven test script with a dataset

Here are some examples of problems that data-driving tests solves:

Problem: During recording, you create a personnel file for a new employee, using the employee's unique identification number. Each time the test is run without data-driving the test, there is an attempt to create the same personnel file and supply the same identification number. The application rejects the duplicate requests.

Solution: You can data-drive the test script to send different employee data, including identification numbers, to the server each time the test is run.

Problem: You delete a record during recording. When you run the test without data-driving the test, Rational® Functional TesterHCL OneTest™ UI attempts to delete the same record, and `Record Not Found` errors result.

Solution: You can data-drive the test script to reference a different record in the deletion request each time the script is played back.

Data-driving tests gives a more accurate picture of the way the application under test works in the real world with real data.

Data-driven functional tests

To data-drive a test script, you need to select the controls or objects in the application-under-test using either the Object Finder Tool method or the Test Object Browser method.

Before you begin

About this task

A dataset can be populated with data from the application. A dataset is a collection of related data records. A dataset supplies data values to the variables in a test script during test script playback.

You can use any of the following methods to select an object or a control:

- **Object Finder Tool** method -- Use this tool to select an object and all descendents of the object, select one object, or select an object and the immediate children of an object.

  Note:

- **Test Object Browser** method -- Use this method to browse for the object that you want to select. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level, Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application. The hierarchical tree is a dynamic view of the currently available objects.

To data drive a test script:

1. Create a functional test project.
2. Start recording a test script.
   a. In the Select Script Assets dialog box, in the **dataset Record Selection Order** box, select one of the following types of dataset record selection orders:
      - **Sequential** -- At playback, the test script accesses records in the dataset in the order that they appear in the dataset.
      - **Random** -- At playback, the test script randomly accesses every record in the dataset once.

   b. Click **Finish**.
   Result
   The Rational® Functional TesterHCL OneTest™ UI window minimizes and the Recording Monitor opens.
3. Start the application you want to test and navigate through the application to the dialog box that you want to data-drive.
   a. Click **Start Application** on the **Recording** toolbar.

   See Starting Your Test Applications on page 896 for more information. (If your application is already running, you do not need to do this step.)

   b. Perform any actions in the application that you want to record in the test script.
4. Data-drive the test.

a. On the **Recording** toolbar, click **Insert Data Driven Commands** ().
Result

The test script recording pauses and the Insert Data Driven Actions page opens.

b. In the application-under-test, type the initial values that you want to see in the dataset in the fields that you want to data drive.

By populating these fields while the recorder pauses, you do not record unnecessary actions in the test script. Any change to the control flow of the program while the test script recording pauses is not recorded in the test script.

5. Under **Populate then Select Test Objects**, choose one of the following methods:
   -  **Press and drag hand to select test objects** -- Use this method to select an object and all the descendents of the selected object. This is the most common and direct method of selecting an object.
   -  **Use selection wizard to select test objects** -- Click to use the **Drag Hand Selection** method with its options, or the **Test Object Browser** method. The Select Object to Data Drive page opens.
6. If you choose  **Press and drag hand to select test objects**, take the following steps:

a. Use the mouse to drag the hand, the Object Finder tool (), to the object in the application that you want to select.
Result

Rational® Functional TesterHCL OneTest™ UI outlines the object with a red border.

b. Release the mouse button.
Result

The Data Drive Actions page opens. In the Data Drive Actions page, under the **DataDriven Commands** table, information appears about the objects you selected.

You can place your mouse pointer over a row in this table to view the line of code that Rational® Functional TesterHCL OneTest™ UI inserts into the test script to data-drive the application-under-test.

7. If you choose  **Use selection wizard to select test objects**, click the **Selection method** arrow to select one of the following methods:

- **Drag Hand Selection** method -- Use this tool to include only the selected object, the selected object and the immediate children of the selected object, or to include the selected object and all descendents of the selected object.

  Note: The **Drag Hand Selection** method is not available on Linux environments such as Ubuntu and Red Hat Enterprise Linux (RHEL). You must use the **Test Object Browser** method on Linux environments.

- **Test Object Browser** method -- Use this method to browse for the object that you want to select.

8. If you choose the **Drag Hand Selection** method, take the following steps:

   a. Optionally, select or clear **After selecting an object advance to the next page**.

   b. Use the mouse to drag the hand, the Object Finder tool ( ), to the object in the application that you want to select.

      Rational® Functional TesterHCL OneTest™ UI outlines the object with a red border.

   c. Release the mouse button.

   d. Click **Next** if you did not select **After selecting an object advance to next page**.

   e. Click one of the following options:
      - **Just the selected object**
      - **Include the immediate children of the selected object**
      - **Include all descendents of the selected object**

9. If you choose **Test Object Browser** method, take the following steps:

   a. Browse the object tree to find the object that you want to data-drive.

   b. Click the part of the tree that you want to select.

   c. Click **Next**.

   d. Choose one of the following options:
      - **Just the selected object**
      - **Include the immediate children of the selected object**
      - **Include all descendents of the selected object**

   e. Click **Finish**.
   Result

   The Insert Data Driven Actions page opens with information filled in under **Data Driven Commands** and **Selected Command Description**.

10. Optionally, in the **Data Driven Commands** table, under the **Variable** header, type a descriptive name for the name of each variable in the dataset.
11. Optionally, in the **Data Driven Commands** table, under the **Initial Value** header, double-click the initial value, and then type in a new initial value or click the arrow to select a new value from the list.

    For example, you can change the initial value of a test object to test the non-default states of an application.
12. Optionally, in the **Data Driven Commands** table, make any of the following changes.
    - Click to move the selected row earlier in the order of execution in the Data Driven Commands table.
    - Click to move the selected row later in the order of execution in the Data Driven Commands table.
    - Click to delete a selected row from the Data Driven Commands table.
    - Click to highlight a test object in the application-under-test. Select a test object in the Data Driven Commands table, and then click this icon.
    - Click to display or hide recognition and administrative properties for a selected test object
13. Click **OK** to finish data-driving the script. The Insert Data Driven Actions page closes and Rational® Functional TesterHCL OneTest™ UI populates a dataset with data collected from the application.
14. To finish recording the test script:

    a. Perform any actions in the application you want to record.

    b. If you want to record a verification point, locate the object in your application you want to test, and then click the **Insert Verification Point or Action Command** button.

    Tip: Click the **Help** button while creating the verification point for more information on the Verification Point and Action Wizard, or see Creating a Properties Verification Point on page 904 for an example of how to create a properties verification point.

    You can use a dataset reference instead of a literal value for the value you are testing in the verification point.

    c. If you want to insert any script support functions into the script, such as a call script command, log entry, timer, script delay command, or comment, click the **Insert Script Support Commands** button.

    Click the **Help** button in the Script Support Functions on page 890 dialog box for information on these functions.

    d. Close your application, if you want closing the application to be part of the script.

    e. On the **Recording** toolbar, click **Stop Recording** ( ) to write all recorded information to the test script and update the dataset with new variables and associated initial values.

Result

The Rational® Functional TesterHCL OneTest™ UI window opens and the script displays in the editor window.

15. You can add data to the dataset after you finish recording the test script. For more information about editing a dataset, see Editing datasets on page 947.

Working with datasets

A dataset is a collection of related data records which supplies data values to the variables in a test script during test script playback.

You can use datasets to supply realist data and to stress an application with a realistic amount of data.

Using Rational® Functional TesterHCL OneTest™ UI, you can create a data-driven test by selecting the controls or objects in an application-under-test to data-drive. Rational® Functional TesterHCL OneTest™ UI creates a dataset in which you can edit and add data. You can use a single test script repeatedly with varying input and response data.

You can use the dataset feature in the following ways:

- To add realistic data to a test script.
- To import data from a dataset or a .csv file created using a spreadsheet application.
- To edit dataset values.
- To change the dataset record selection order to determine how the test script accesses an associated dataset during playback.

While working with simplified test scripts, you can either create a data-driven test script during recording or you can insert data-driven commands into the simplified test script by using the application visuals. You can also create multiple datasets for a script and associate a dataset to a group in a simplified script.

Private and shared datasets

Every test script that you create has a private test dataset associated with it. The initial private test dataset is a placeholder and is empty until you data-drive a test script, or add new data to it.

You can create a shared dataset by creating a new dataset, or you can associate a dataset with several test scripts to share a dataset.

Creating a dataset

A dataset is a collection of related data records which supplies realistic data values to the variables in a test script during test script playback. You can create data from scratch or import existing data into a new dataset from another dataset or a .csv file.

1. Click **Create a Test dataset** (). The **Create New Dataset Location and Filename** dialog box is displayed.
2. Click the project that must contain the dataset. The selected project is displayed in the **Enter, create, or select the parent folder** field.
3. In the **Name** field, enter the required name of the dataset, and click **Next**.
4. Optional: In the **Description**, enter the required description for the dataset.
5. In the **Dimensions** field, specify the number of rows and columns for the dataset.
6. Click **Finish**. The new dataset is displayed in a browser.

Importing to datasets

You can import test data into a dataset using a comma-separated value (CSV) file that enables you to import large volumes of test data to a dataset rather than manually entering them.

Before you begin

You must have created a test that contains a dataset with at least one column.

1. Under the Functional Test Projects view, double-click the dataset. The dataset is displayed in a browser.
2. Click **Import**.
3. Choose the .csv file to import and click **Open**.
4. You can choose the following options in the Import dialog box:
   Choose from:
   - Click **Append** to add rows and columns from the selected CSV file to the end of the dataset.
   - Click **Overwrite** to add the rows and columns from the selected CSV file from the beginning of the dataset.
5. Click **OK**. Based on your selection, the rows and columns values are added either to the end of the dataset or from the beginning of the dataset.
6. Click **Save** and close the browser. Click **Discard Changes** if you do not want to save the changes.

Results

The data from the CSV file is imported into the dataset.

Inserting data-driven commands into a script by using an application visual

You can add data-driven commands to the script by using an application visual.

Before you begin

Prerequisite: To use the application visuals for inserting data-drive commands into a script, the **Insert Data Driven Commands** feature must be enabled in the **Preferences** page before recording the script.

1. Select the test line in the script editor that refers to the required application visual of the test application. The application visual that contains the control is displayed.

Note: The thumbnail pane in the Application view displays images of all the application visuals that are captured for the project. You can select the required application visual image in the thumbnail pane and view the application visual in the Application view.

2. In the Application view, move the mouse pointer over the control to use for the data-driven command.
   Result
   By default, the control is highlighted in red.
3. Right-click and select **Insert Data Driven Command**.
   Result
   The data driven command for the control is inserted into the test script. You can drag the test line in the script editor to arrange the test lines in the required sequence for playback.
4. In the **Test dataset** view, add the data for the control.
5. Click **File > Save** to save the modified test script.
   During script playback, the data for the control is retrieved from the dataset.

Associating a dataset with a group in a simplified script

You can associate a dataset with a group in a simplified script. With this capability, you can create multiple datasets for a script and associate the datasets with the groups in a script.

1. Select a group in the simplified script editor that contains the test lines to be data-driven.
   Result
   The **General** page in the **Properties** view displays the dataset section.
2. On the **General** page of the **Properties** view, you must specify the dataset details:
   a. To create a dataset and associate it with the group, type a new dataset name. If a dataset already exists, select the dataset from the list.
   b. Type the dataset iteration count to specify the number of times the test lines in the group must be run by accessing the records in the dataset.
3. Click **Script > Find Literals and Replace with Dataset Reference** to replace the existing literals in the group with the dataset reference.
   Ensure that you add the literals to the dataset while replacing the literals with the dataset reference so that the values are added to the dataset. For more information, see the topic about replacing literals with dataset reference.
4. Click **File > Save** to save the test script.
   The Test dataset view displays the newly associated dataset with the literal values as records. You can add more data to the dataset in the Test dataset view.

Note: If you insert data-driven commands for other controls in the group by using an application visual, the data for the control is automatically added to the dataset that is associated with the group.

Result

During script playback, the data for the controls is retrieved from the associated dataset.

Encrypting datasets

To secure test data, you must encrypt datasets. You can encrypt one or more columns of a dataset using a password. You are prompted to enter your password when you run a test that utilizes dataset with encrypted variables.

Before you begin

You must have created a test that contains a dataset.

1. Under the Functional Test Projects view, double-click the required dataset. The dataset is displayed in a browser.
2. Right-click the heading of the column and select **Encrypt column**.
   Result
   The Encrypt dataset dialog box is displayed.
3. Enter the required password to encrypt the column and confirm the password.

   Notes:
   - If you forget the password to a dataset, there is no way to recover it.
   - If you have already encrypted other columns in the dataset, you must enter the password that you used previously. You can use only one password to encrypt columns in a dataset.

4. Click **OK**.
   Result
   Asterisks are displayed instead of actual values of the encrypted variables.

Results

The dataset column is encrypted.

Changing passwords on encrypted datasets

To secure test data, you must encrypt datasets. All the columns in a dataset are encrypted using a single password. You can change this password at any instance.

Before you begin

You must have created a test that contains a dataset with at least one encrypted column.

1. Under Functional Test Projects view, double-click the required dataset. The dataset is displayed in a browser.
2. Right-click the heading of the column and select **Change password**. The Change password dialog box is displayed.
3. Enter the old password and the new password, and confirm the new password.
4. Click **OK**.

Results

The password on the encrypted dataset is changed.

Decrypting Datasets

You must decrypt the dataset if you want to export a dataset into a .csv file. If you decrypt a dataset, it revokes the protection offered to the test data in it.

Before you begin

You must have created a test that contains a dataset with at least one encrypted column.

1. Under the Functional Test Projects view, double-click the required dataset. The dataset is displayed in a browser.
2. Right-click the heading of the column from which you want to remove encryption and select **Decrypt column**.
   Result
   The Decrypt column dialog box is displayed.
3. Enter the password that you used to encrypt the column.
4. Click **Ok**.

Results

The encryption is removed and the dataset column is decrypted.

Editing datasets

After you data-drive a test to create a dataset or create an empty dataset, you can edit the records and variables in the dataset.

About this task

A dataset is a test dataset, a collection of related data records which supplies data values to the variables in a test script during test script playback. A record is a row in a dataset. A variable is a column in a dataset.

You can make the following changes to a dataset:

- Add, remove, move, or edit a row
- Add, remove, move, or edit a column
- Edit or clear cell(s)
- Cut, copy, or paste a cell, row, or column

Note: To have seamless access to a dataset CSV editor, you can use any one of the following web browsers on Windows, Linux or Mac operating systems:

- Mozilla Firefox
- Google Chrome
- Microsoft Edge based on Chromium

Selecting a record

To select a record, you must click the column number. For example, 0, 1, or 2.

Adding a record

1. Click anywhere in the dataset editor or select a record, right-click, and then click **Add Record**.
   Result

   The new record appears after the selected record unless you select the first row.
2. If you select the first row, click **Add Before** or **Add After** to place the new record before or after the first record, and then click **OK**.

Removing a record

1. Select a record that you want to delete.
2. Click **Remove Record**.

Moving a record

To move a record before or after another record:

1. Select a record, and then right-click **Edit Record**.
2. Click the **Index** arrow to select the location where you want to move the record.

   For example, select **Before 0** to move a record before record 0 or select **After 12** to move a record after record 12.
3. Click **OK**.

Editing dataset values

1. Select the cell you want to change.
2. Double-click the selected cell and type the new value of the cell.

Adding variables

1. Select a cell or click anywhere in the dataset editor, and then right-click **Add Variable**.
2. In the Add Variable dialog box, type the name of the new variable.
3. Type the full class name for the variable.

   The system String class is the default.
4. Click the **Add** arrow to select the location where you want to place the new variable.

   For example, select **Before *NameofVariable*** to place the new variable to the before an existing variable or select **After *NameofVariable*** to place the new variable after an existing variable.
5. Click **OK**.

Removing variables

1. Right-click a cell in the variable you want to remove.
2. Click **Remove Variable**.

Changing names, types, or move variables

1. Select a cell in the variable that you want to change. Select the variable header that you want to change.
2. Right-click, and then click **Edit Variable**.
3. In the Edit Variable dialog box, double-click or select the name of the variable and then type the new name of the variable.
4. Type the full class name for the variable.
   The system String class is the default.
5. Click the **Move** arrow to select the location where you want to move the variable.

   For example, select **Before *NameofVariable*** to move the variable before an existing variable or select **After *NameofVariable*** to move the variable after an existing variable in the dataset.
6. Click **OK**.

Cutting, copying or pasting cells, records, or variables

To cut, copy, or paste a cell, a record, or a variable:

- To delete data in a cell, record, or variable to the clipboard and copy it to the clipboard, select a cell, a record, or a variable, right-click, and then click **Cut**.
- To paste the contents of the clipboard, select a cell, record, or variable that you want to overwrite with the contents of the clipboard, right-click, and then click **Paste**.
- To copy a cell, record, or variable to the clipboard, select a cell, a record, or a variable, right-click, and then click **Copy**.

Updating dataset values programmatically

You can also update the dataset values by using the getValue and setValue APIs.

Use the following code sample to update the dataset in a test script from the required CSV file.

```
public void testMain(Object[] args)
    {
        String cursorId= "differentTestCursor";
        Cursor cursorOptions = Cursor.builder()
                .cursorId(cursorId)
                 .accessMode(AccessModeEnum.OVERWRITE)
                 .fetchMode(FetchModeEnum.SEQUENTIAL)
                 .shareMode(ShareModeEnum.SHARED)
                 .lastActive(null)
                 .wrap(true)
                 .build();

        //provide name of the dataset file as argument
        Path path= Paths.get("setCellGetCell.csv");
        String DatasetPath=path.toAbsolutePath().toString();
        String DatasetEncodedPath = Base64.getEncoder().encodeToString(DatasetPath.getBytes());

        DatasetCursor createdCursor= CursorFactory.getWorkbenchDatasetCursor("localhost", 7081,
                DatasetEncodedPath, cursorOptions);

        DatasetRow row= createdCursor.nextRow();
        String ExistigValue=row.getValue("Column1");

        System.out.println(ExistigValue);
        logInfo(ExistigValue);

        row.setValue("Column1", "Test");
        String Newvalue=row.getValue("Column1");
        System.out.println(Newvalue);
        logInfo(Newvalue);

        createdCursor.deleteCursor(true);
        }
}
```

Replacing literals with dataset references

If you have an existing test script, you can replace literal values in the script with dataset references to add realistic data to the script. You can find and replace all literals, or just the number, string, or boolean literals with a dataset reference.

About this task

You can also add a literal from a script to a dataset. If you do not use an existing dataset variable, the same literal values (the values that were captured when you recorded the test script) are used each time you run the script.

1. Click **Script > Find Literals and Replace with Dataset Reference**.
2. Under **Direction**, click **Forward** or **Backward** to set the direction you want to search through a test script.

3. Under **Literal Type**, set the type of literal you want to find from the following list:
   Choose from:
   - **All** -- Click to find all literals in a script.
   - **Numbers** -- Click to find number literals in a script. A number includes integers (a whole number, not a fractional number, that can be positive, negative, or zero) or floating numbers (positive and negative decimal numbers).
   - **Strings** -- Click to find string literals in a script. A string stores alphanumeric values such as name, city, or state.
   - **Booleans** -- Click to find boolean literals in a script. Any use of the boolean literals true or false are flagged for substitution.
4. Click **Find** to start the search.

   The name of the literal found in the test script appears under **Literal**.
5. Click the **Dataset Variable** arrow to display the dataset variables and then click the dataset variable that you want the script to reference.
6. Optionally, type a new variable name for the dataset variable you selected.
7. Optionally, click the **Add Literal to Dataset** check box to add the literal value to the dataset.
8. Click **Replace** to replace the literal in the script with a dataset variable reference.

   The cursor automatically moves to the next literal in the test script.

   Note: If you have a literal that is a choice in a combo box in the application-under-test, when you replace the literal with a dataset reference, you get a string data type instead of an a enumerator data type in the dataset.

9. When you find the next literal in the test script, repeat steps 5 through 7.
10. When finished, click **Close**.
    Result

    Rational® Functional TesterHCL OneTest™ UI updates the dataset with any new columns and initial values that you add in this dialog box.

Dataset references and verification points

When you create a verification point with a dataset reference, you can supply variable data to make your tests more realistic.

Each time you play back a script with an associated dataset, the script accesses one record in the dataset. The verification point uses the dataset reference to access a variable in that record. At playback, Functional Test substitutes the variable in the dataset for the dataset reference and compares it to the actual results.

The log contains the record of events that occur while playing back a script and the results of all verification points executed. Actual test results (with the dataset reference resolved) that vary from the baseline results are defects or intentional changes in the application.

In the following diagram, the left box represents typical script assets:

(1) Script with an associated dataset

(2) dataset

(3) Object map

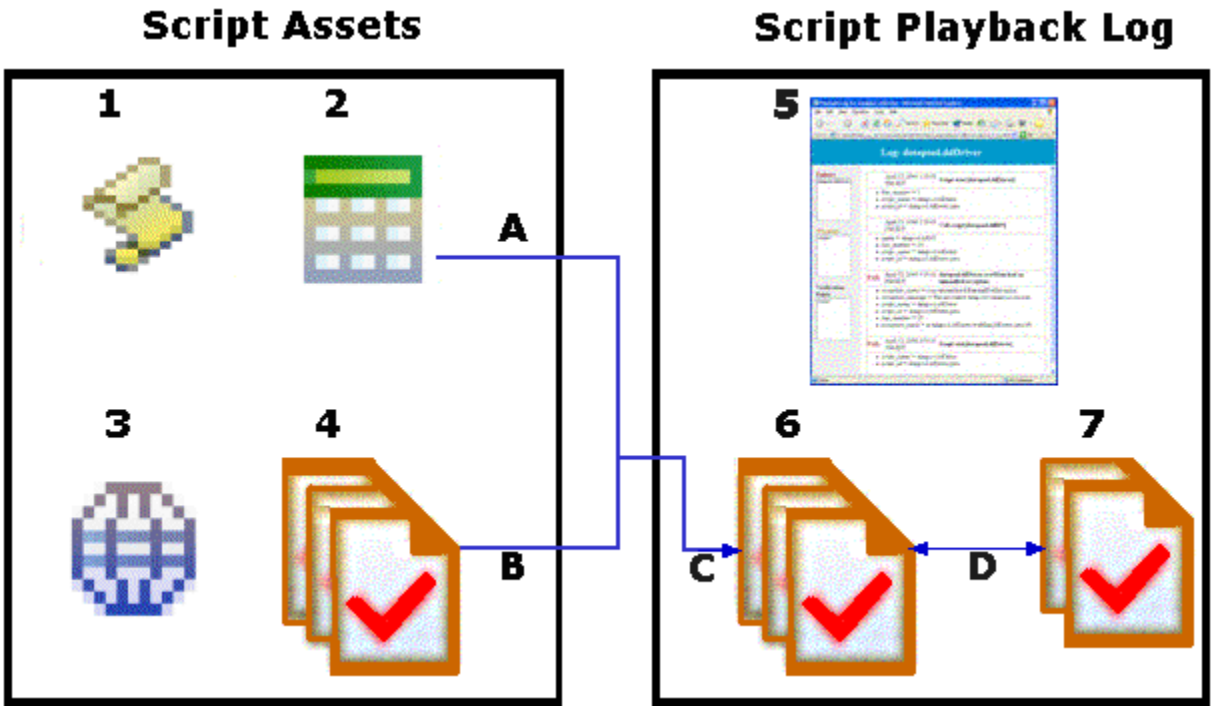
(4) Verification point baseline -- A baseline is the captured data maintained with a script. The verification point, in this example, has a dataset reference.

The right box represents the following:

(5) Test log that records the verification points that passed or failed

(6) Expected data, equal to the script's baseline data, with dataset references resolved to the values in the active record at the time the verification test was performed.

(7) Actual data is the data from the software-under-test at the time the verification point is performed.



When you run a script the following events take place:

(A) The verification point accesses the data in the dataset creating the expected result from the baseline and the active dataset record.

(B) The dataset reference resolves and supplies data to the verification point.

(C) The expected result is compared to the actual result.

(D) Expected and actual results are recorded in the log. Any deviation from the expected results are logged as a failure in the test log.

Changing the dataset record selection order

The test dataset record selection order determines how a test script accesses records in the test script's associated dataset when you play back the test script.

1. In the Projects viewSolution Explorer, select the test script that is associated with the dataset that you want to change.
2. Right-click the selected test script and click **Properties**.
3. Click **Functional Test Script**.
   The Functional Test Script Properties page opens. Rational® Functional TesterHCL OneTest™ UI uses the test dataset for the test script you selected.
4. To change the dataset record selection order, click **Dataset Record Selection Order** , and then select the following record selection order:
   - **Sequential**: You can use this option to make a test script access records in the dataset in the order that they appear in the dataset. The sequential record selection order is the default dataset record selection order.
   - **Random**: You can use this option to make a test script access records in the dataset randomly. A random record selection order accesses every record in the dataset once.
5. Click **Apply**.

Results

The dataset record selection order is modified.

Associating a dataset with a test script

You can associate a test script with a dataset to use external data to drive the application instead of using a literal value.

About this task

A dataset is a test dataset, a collection of related data records. It supplies data values to the variables in a test script during test script playback.

1. In the Projects view, select a dataset. In the Solution Explorer, select a dataset.
2. Right-click the selected dataset and click **Associate with Script**. The Associate the dataset with scripts dialog box is displayed.
3. Expand the project to open the list of scripts.
4. Click one or more test scripts to associate with the dataset that you selected.
5. Click **Finish**.

 **CAUTION:** If you change the dataset associated with a script, the script may run incorrectly.

6. If you receive a message that a script is already associated with a dataset, you must complete one of the following steps:
   Choose from:
   - Click **Yes** to change the dataset associated with the script to another dataset.
   - Click **No** to keep the dataset associated with the script.

Associating a test script with a dataset

You can associate a test script with a dataset to use external data to drive the application instead of using a literal value.

About this task

A dataset is a test dataset, a collection of related data records. It supplies data values to the variables in a test script during test script playback.

1. Open the test script that you want to change.
2. From the Script Explorer, select **Test Dataset**.
3. Right-click and select **Associate with Dataset**.
4. Type the name and path of the test dataset or select a dataset from the list of datasets.
5. Click **OK**.

Removing a dataset association

You can remove dataset association with a test script.

1. Open the test script you want to change.
2. In the Script Explorer, right-click the Test dataset directory and click **Remove Dataset Association**.
3. You must complete one of the following steps:
   Choose from:
   - Click **Yes** to remove the dataset association.
   - Click **No** to keep the current dataset association.

Deleting a dataset

You can delete a dataset if you no longer require it.

Before you begin

If you want to delete a dataset, you must remove all the associations of the required dataset with the test scripts.

1. In the Projects view, select the dataset you want to deleteIn the Solution Explorer, select the dataset you want to delete.
2. Right-click the selected dataset and click **Delete**. The Delete dialog box is displayed.
3. Click **Yes** to confirm that you want to delete the dataset.

Results

The dataset is now deleted.

Replacing literals with dataset references in the Visual Studio IDE

If you have an existing test script, you can replace literal values in the script with dataset references to add realistic data to the script. You can find and replace all literals, or selectively replace the number, string, or boolean literals with a dataset reference.

Before you begin

About this task

You can also add a literal from a script to a dataset. If you do not use an existing dataset variable, the same literal values (the values that were captured when you recorded the test script) are used each time that you run the script.

To replace a literal with a dataset reference:

1. Click **Script > Find Literals and Replace with dataset Reference**.
2. Under **Literal Type**, select or clear the appropriate check boxes:
    - **All** -- Click to find all literals in a script.
        - **Numbers** -- Click to find number literals in a script. A number includes integers (a whole number, not a fractional number, that can be positive, negative, or zero)or floating numbers (positive and negative decimal numbers).
        - **Strings** -- Click to find string literals in a script. A string stores alphanumeric values such as name, city, or state.
        - **Booleans** -- Click to find boolean literals in a script. Any use of the boolean literals true or false are flagged for substitution.
    - **Search up** -- Click to move back up through the test script. Clear to move forward through the test script.
3. Click **Find Next** to start the search.

   The name of the literal found in the test script appears under **Literal**.
4. Click the **dataset Variable** arrow to display the dataset variables and then click the dataset variable that you want the script to reference.
5. Optionally, type a new variable name for the dataset variable you selected.
6. Optionally, click the **Add literal to dataset** check box to add the literal value to the dataset.
7. Click **Replace** to replace the literal in the script with a dataset variable reference.

   Result

The cursor automatically moves to the next literal in the test script.

Note: If you have a literal that is a choice in a combo box in the application-under-test, when you replace the literal with a dataset reference, you get a string data type instead of an a enumerator data type in the dataset.

8. When you find the next literal in the test script, repeat steps 4 through 6.
9. When finished, click **Close**.

   Result

   Rational® Functional TesterHCL OneTest™ UI updates the dataset with any new columns and initial values that you add in this dialog box.

Results

Related Topics:

Editing datasets on page 947

Associating a dataset with a Test Script on page 953

dataset Literal Substitution Dialog Box on page 1899

What to do next

Testing terminal-based applications

Use the Rational® Functional TesterHCL OneTest™ UI Terminal-based Applications feature to create test scripts and automate your host-application test cases. You can test host attributes, host field attributes, and screen-flow through a host application. It uses terminal verification points and properties, as well as synchronization code to identify the readiness of terminal for user input.

You can use the terminal-based applications feature to perform the following tasks:

- Store, load, and share common host configurations by using a properties file. The connection configuration can be loaded automatically through scripts, using these files.
- Record or play back scripts against multiple host terminals.
- Start the terminal even when you are not recording or playing back your scripts. With this function, you can interact with the host without leaving the working Eclipse environment.
- Perform data driven testing.

For information about the Host Access Class Library (HACL) for Java APIs, see IBM WebSphere Host On-Demand Information Centerl

Importing certificates from the server for secured connections

Starting from Rational® Functional TesterHCL OneTest™ UI 9.1.1, you can import certificates from the server to connect to the host machines securely. You can create a `CustomizedCAs.p12 / CustomizedCAs.jks` keystore through the Extension for Terminal-based Applications that produces the *.p12/JKS file. This file passes the host server's self-signed certificate credentials to the terminal to allow a secure connection.

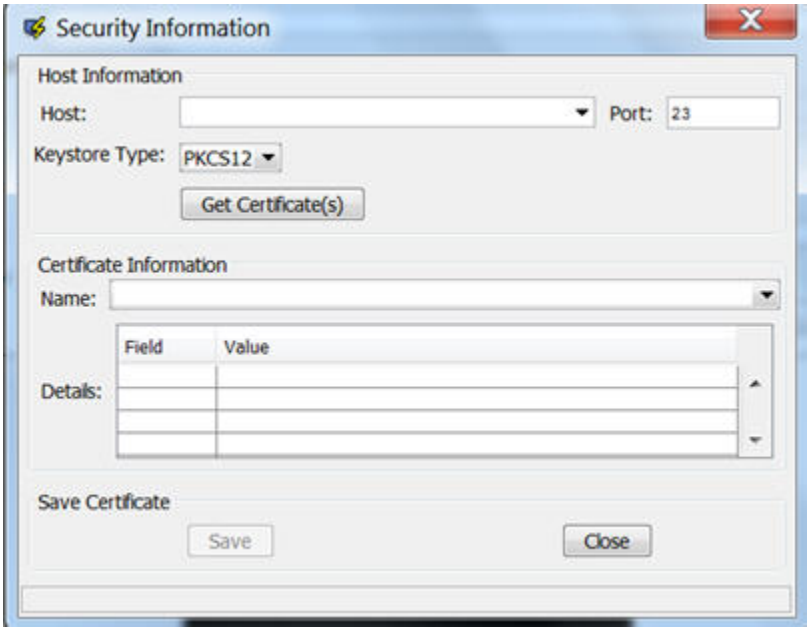
About this task

Prior to version 9.1.1, you could use SSL to connect to the host machine securely. See Using SSL to connect to host machines on page 1036.

1. Start the Extension for Terminal based Applications by clicking the Launch button in Rational® Functional TesterHCL OneTest™ UI.



2. Click **Session > Security** to open the Security Information window.



3. Select or type the Host address and Port number.
4. Select the type of keystore where the certificate can be saved. Depending on the connection protocol supported by the host, you can choose **PKCS12** or **JKS**.
5. Click the **Get Certificate(s)** button to retrieve the certificates from the host.
6. After the certificates are retrieved, click the **Save** button to save the extracted certificate to the appropriate keystore (`CustomizedCAs.p12` or `CustomizedCAs.jks`).

7. Click the Status bar to open the location where the keystore is created with the certificate. This location would be `C:\Users\<current user>\Application Data\ibm\RFT\Extension for Terminal-based Applications` on a Windows™ machine.
8. Copy the `.p12` or `.jks` file to the `<IBMIMShared\plugins>\com.ibm.test.terminal_8.5.0.vXXXX` `<HCLIMShared\plugins>\com.ibm.test.terminal_8.5.0.vXXXX` folder. This plugin folder also includes the `terminal.jar` and `TerminalTester.jar` files.
9. Close the Security Information window and restart the Extension for Terminal-based Applications.
10. Type the Host address, Port number, and terminal type information and click the **Advanced Settings** button.
11. Set the properties depending on the type of certificate.
    - For `CustomizedCAs.p12`, you must set the following properties:.

Property Name	Set the value...
SSL	*true*
SSLTelnetNegotiated	*true*

    - For `CustomizedCAs.jks`, you must set the following properties:

Property Name	Set the value...
sslUseJSSE	*true*
sslJSSETrustStore	Provide the full path of `CustomizedCAs.jsk`. For example, `C:\Program Files\IBM\IBMIMShared\plugins\com.ibm.test.terminal_8.5.0.v20170703_0428\CustomizedCAs.jksC:\Program Files\HCL\HCLIMShared\plugins\com.ibm.test.terminal_8.5.0.v20170703_0428\CustomizedCAs.jks`
sslJSSETrustStorePassword	*hodpwd*
tlsProtocolVersion	*TLSv1.2* **Note:** If the host supports an older version of the protocol, the application will fall back to the older version.
sslJSSETrustStoreType	*jks*

Property Name	Set the value...
SSL	*true*
SSLTelnetNegotiated	*true*

Note: You must set SSLTelnetNegotiated to *true* only when you connect to a Telnet server that supports IETF Internet-Draft TLS-based Telnet Security. The Internet-Draft defines the protocol for performing the SSL Handshake over a Telnet connection.

Creating a host connection script

You can create a host connection script so that you can interact with the host session, navigate to other screens, and create data verification points.

1. Click the **Record a Functional Test Script** icon on the toolbar to start recording a new script.
2. Provide a name for the script, and click **Finish**.
   Result
   The Rational® Functional TesterHCL OneTest™ UI window minimizes, and the **Recording Monitor** opens.
3. Click the **Start Application** icon on the **Recording Monitor** toolbar.
4. Select **Extension for Terminal Based Applications** from the list, and click **OK**.
5. Specify the basic connection properties. For more information about basic connection properties, see the related topics.
6. **Optional:** Click **Advanced**, and type the properties in the Advanced Settings window. For more information about advanced connection properties see the related topics.
7. Click the **Open a Connection Properties File** icon on the tool bar.
8. Select the file name, and click **OK**.
9. Click the **Connect using the current connection properties** icon on the tool bar.

Note: If you type invalid properties or leave any required properties unspecified, an error message is displayed. Correct the invalid entries in Extension for Terminal-based Applications window, and connect.

What to do next

After the session is connected, you can interact with the host session, navigate to other screens, and create data verification points.

You can also create scripts that start from a non-login host screen:

1. Start Extension for Terminal-based Applications.
2. Log in to your host session.
3. Navigate to the location where you want to start recording the script.
4. Start recording your script.

Creating a new connection configuration file

Before connecting to your host session, you can create a new connection configuration by saving the connection values in a host connection configuration file.

1. Click the **Save the Connection Properties** icon or **Save the connection properties to a file** icon on the tool bar.
2. Type a file name.

   Choose from:

   - If you type an invalid file name, such as one that includes restricted characters or reserved operating system keywords, an error message is displayed with the option of changing the file name or canceling the operation.
   - If you type a file name that already exists, a warning message is displayed with the option of writing over the existing file, changing the file name, or canceling the operation.
3. Click **OK**.

   Result

   The connection configuration file is saved in the directory specified in the **Default folder for Connection Configuration Files** field in Preferences window.
4. Click **Connection Configurations** list.

   Result

   The **Connection Configurations** field lists all the .conn files that are present in the default folder that is specified in **Default folder for Connection Configuration Files** field in the Preferences window.

   Note: If you have used the **Open** function to load a .conn file from a non-default folder, the last 10 opened configuration files are displayed in addition to the .conn files that are located in the default directory.

Saving connection properties

You can save your connection properties for later use. The connection properties are saved with a .conn extension in location specified in the Preferences window. On loading the connection properties file, the current properties are replaced with the properties stored in the file.

1. Create a new connection configuration file. For information on creating a new connection configuration file, see related topics.
2. Click the **Save the Connection Properties** or icon or **Save the connection properties to a file** icon on the tool bar.

The connection properties are saved in a `.conn` file in the specified location as **Default Connection Properties Location** in Preferences window. The default location also specifies the location of the properties files list when you are loading a properties file.

Note: Loading the saved properties file replaces the current properties with those stored in the file.

3. Type a file name.
4. Click **OK**.

Modifying invalid preferences

On starting Extension for Terminal-based Applications, an error message might be displayed for each invalid preference specification for Timeout, OIA State Timeout, Polling Interval, and Minimum time to wait. You can correct these invalid settings.

1. Note the preference in error, and click **OK** for each error message.
2. From menu bar, click **Sessions > Preferences**.
3. Modify the invalid preference setting that is reported in the message.
4. Click **OK** after modifying the preference setting.

Creating scripts using multiple terminals

You can create scripts that simultaneously automate the testing of multiple host terminals. Using this feature, you can create scripts with host applications that react to the interactions with other host applications.

1. Start recording a new script using the host connection. For information on recording a host connection script, see related topics.
2. Log in to your host session and interact with the host, navigating to other screens and creating data verification points.
3. To open another terminal session, perform the following steps:

    a. On the Recording Monitor window toolbar, click the **Start Application** icon.
    Result
    The Start Application window opens.

    b. Click **Extension for Terminal-based Applications** from the **Application Name** list, and click **OK**.

4. You can now switch back and forth between the terminals, continually interacting with the hosts, and creating verification points on the different terminals.
5. Log off from both hosts, and on the tool bar click the **Stop recording** icon.
6. Save your script.

What to do next

Note: Extension for Terminal-based Applications uses .conn files to differentiate the sessions. The terminal sessions cannot use the same .conn file in order to distinguish the terminals from one another. Otherwise, when the script is played back, Rational® Functional TesterHCL OneTest™ UI will not recognize that there are two terminal sessions. All actions against either terminal will be played back on a single terminal.

If during playing back your script, any of the verification points fail, then edit your script or object map.

Customizing screen size when connecting to a TN3270 or TN3270E host

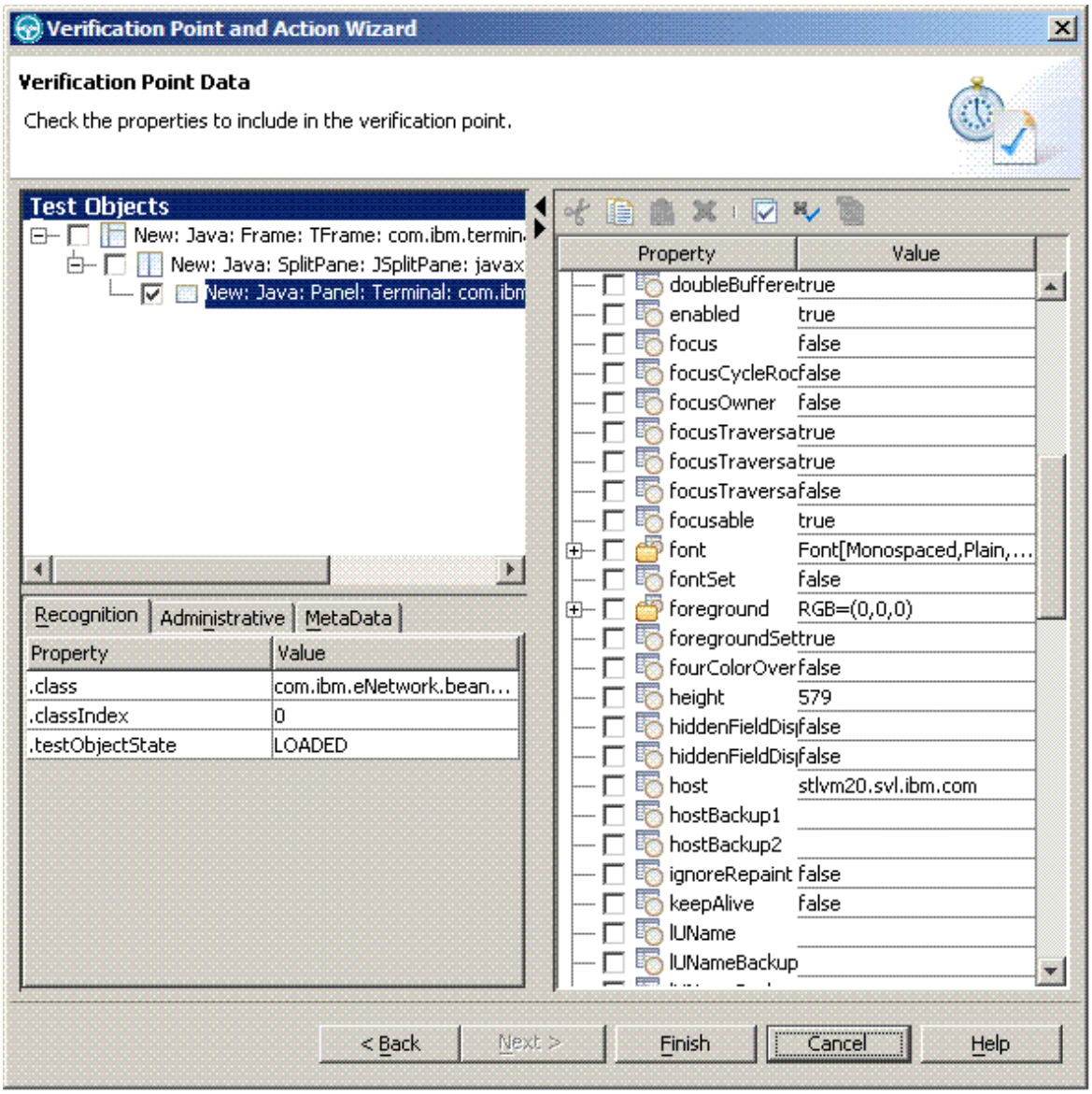
You can customize your screen size when connecting to a TN3270 or TN3270E session. Other types of sessions, such as TN5250 and VT sessions, are not affected by this feature.

1. Select **TN3270** or **TN3270E** from **Terminal Type** list.
2. Select **Custom RowXCol** from **Screen Size** list.
3. Select a valid RowXCol from **RowXCol** field. A valid RowXCol string is comprised of 1-to-3-digit number of rows, optional blanks, the character 'x' or 'X', optional blanks, and a 1-to-3-digit number of columns.
4. Specify connection details.
5. Click **Connect**.

Recording a host connection script

You can automate the testing of a host application by recording a script that connects to, and interacts with a host system. The session is recorded as a set of commands and can be played back at a later time. This allows you to automate the navigation to specific screens.

1. Log in and interact with the host when you create a host connection script.
2. In the window where you want to perform a test, on the Recording Monitor toolbar, click the **Verification Point and Action Wizard** icon.
   Result
3. Drag the object finder over the host terminal (a thick red line outlines the selected terminal or field). You can use the selector to select the entire terminal or any of the fields in the terminal session, and click **Next**.
4. Select the verification values to perform using **Data Verification Point** tab of the Perform an Action page.
   Result
5. Select the required data values and properties
6. Select **All field properties** from the list if you are checking host field attributes. Provide a verification point name, and click **Next**.
   Result
   The Verification Point Data page lists all the host fields from the current host screen.



The fields are named Field_row_col_text, where row is the starting row of the field, col is the starting column of the field and text is the text found in the field.

7. **Optional:** Clear the check box of the field for which you do not want to perform a host-attributes check. By default all host field attributes (except the host field names) are selected by this verification point. If you do not want to test all host field attributes, double-click the item in the list, and clear the attributes in the **Test Data Element** list.

   Note: Avoid checking the name attribute of the host field in the verification point. This attribute might cause problems during script play back.

8. Log off, and click the **Disconnect** icon after creating the verification points.

9. Click the **Stop Recording** icon on the recording monitor toolbar.
   Result
   This stops the recording monitor and generates the script.

Verification points

Verification points are used to test properties of application windows or fields.

You can create verification points using the object finder during recording to select screen or field objects in the host terminal. Create a verification point and check the list of properties that you want to verify. You can verify the following properties and more: location, size, field count, and connection type for screens and color, text, location, size, and numeric for fields.

Creating verification points

You can create verification points to test application objects. Verification points are used to verify that the property values is as expected. When you create a verification point, you are capturing information about an object in the application to establish this as baseline information for comparison during playback.

1. On the Recording Monitor toolbar, click the **Verification Point and Action Wizard** icon to perform a test on a window when recording a script.
2. Select the object that you want to test by dragging the **Object Finder** icon over the part of the host terminal window to test.
   Result
   A thick red line outlines the object that is selected for testing.
3. Click **Next**.
   Result
   The Select an Action window opens.
4. Select an action:
   Choose from:
   - **Perform Data Verification Point**: Tests the data that the object contains. It tests the text in the field.
   - **Perform Properties Verification Point**: Tests one or more properties of the object such as whether a field is protected.
   - **Get a Specific Property Value**: Assigns the value of a specific property of an object to a variable in the test script.
   - **Wait for Selected Test Object**: Causes execution of the test script to wait until the object exists before continuing.
5. Click **Next**. Follow the dialog box and provide values or accept the default values. Depending on the action you select, you might have to complete more than one panel before the final window.
6. Click **Finish** to close the dialog and return to recording a script.

Creating data verification points

You can create data verification points to test the data in your application. When you record the verification point, a baseline of the data is created. Every time you play back the script, the data is compared to check if any changes have occurred. This helps in identifying any mismatch in data.

1. In the Insert Verification Point Data Command window, specify the verification point data.
2. Select a value to create a verification point to verify properties of the fields as well as the screen. The following values are available:
   Choose from:
   - **All field properties**: Checks host field attributes, such as text, highlight, underline, blink, start or end of row or column.
   - **Non-static field values**: Checks the text values on the current host screen by fields that cannot be modified.
3. Check the text values in the current host window by fields that cannot be modified. Select one of the following choices:
   Choose from:
   - **Select One Property Verification Point**: Enables checking only one property of the window.
   - **Data Verification Point**: Provides a quick and easy way of creating a verification point for often tested data such as OIA information and field text.
4. Use the text recognition property with regular expressions to increase the flexibility of your script when defining a screen. The recognition properties are located as children of a specific window. For more information about the text recognition property, see the related topics.
5. Playback the script.



Note: While playing back the script, if any of the verification points fail, then edit the script or object map.

What to do next

When creating verification points for fields, you might encounter fields that are marked as `hidden` (the hidden attribute is set to `true`) that contain text. This text might be hidden on the host window (emulator) based on the login authority. If you create a field data verification point for a hidden field, even though the field is hidden and is not displayed on the host window, you will see the text that is in the field in field properties.

Creating properties verification points

You can create property verification points to test the properties of an object in your application. When you record the verification point, a baseline of the property is created. Every time you play back the script, the property is compared to check if any changes have occurred. This helps in identifying any mismatch in property.

1. Drag the **Object Finder**  icon over an object, and select **Perform Properties Verification Point**. The Insert Properties Verification Point Command window opens.
2. Click **Include Children** to include the immediate children of the object or all its children.

3. Type a name for the verification point in the **Verification Point Name** field.
   Make sure that the name is different from the default name, which makes it easier to locate in the test object map.
4. Click **Next** to select the properties for verification points that you want to test.
5. Click **Finish** to close the Verification Point and Action Wizard and return to recording your script.

Properties for verification points

When you create verification points, the **Verification Point and Action Wizard - Perform an Action** window displays only the properties that apply to the selected component.

Character property verification points

You can create character property verification points to test the character in your application. When you record the verification point, a baseline of the character is created. Every time you play back the script, the character is compared to check if any changes have occurred. This helps in identifying any mismatch in character.

Table 1 shows the properties, descriptions, and default values of character property verification points.

Table 13. Properties for Character property verification points

Property	Description	Default value
background	Specifies the background color of the field.	
position	Specifies the position from the start of the field.	
foreground	Specifies the foreground color of the field.	`false`
reverse	Specifies whether the host field is displayed in reverse video (switch foreground and background colors). This property is not valid for Virtual Terminal (VT) sessions.	`false`
startCol	Specifies the first column of the field.	
startRow	Specifies the first row of the field.	
char	Specifies the current character within the host field.	
blink	Specifies whether the text in the field is flashing. This property is not valid for VT sessions.	`false`
underline	Specifies whether the host field is underlined. This property is not valid for VT sessions.	

Row property verification points

You can create row property verification points to test a row in your application. When you record the verification point, a baseline of the row is created. Every time you play back the script, the row is compared to check if any changes have occurred. This helps in identifying any mismatch in rows.

Table 1 shows the properties, descriptions, and default values of row property verification points.

Table 14. Properties for row property verification points

Property	Description	Default value
char	Specifies the text of the current row.	
class	Specifies the class of the row.	com.ibm.eNetwork.ECL.E-CLPSUpdate
length	Specifies the length of the row.	
rowEnd	Specifies the last pixel position of the row.	
rowStart	Specifies the first pixel position of the row.	
screenCols	Specifies the number of columns in the screen.	
screenRows	Specify the number of rows in the screen.	
startRow	Specifies the current row.	
updatedLength	Specifies the length of the row.	

Properties of field property verification points

You can create field property verification points to test the fields in your application. When you record the verification point, a baseline of the field is created. Every time you play back the script, the field is compared to check if any changes have occurred. This helps in identifying any mismatch in field.

Table 1 shows the properties, descriptions, and default values of field property verification points.

Table 15. Field property verification points properties

Property	Description	Default value
background	Specifies the background color of the field.	
blink	Specifies whether the text in the field is flashing.	`false`
class	Specifies the Rational® Functional TesterHCL OneTest™ UI class name. For example, `HtmlTable` is the class name for a <Table> element.	`com.ibm.eNetwork.ECL.E-CLField`
endCol	Specifies the last column of the field.	
endRow	Specifies the last row of the field.	
foreground	Specifies the foreground color of the field.	`java,awt,Col-or[r=0,g=0,b=0]`
hidden	Specifies whether the host field is hidden.	
highIntensity	Specifies whether the host field uses high intensity colors.	

Table 15. Field property verification points properties (continued)

Property	Description	Default value
length	Specifies the length of the host field (number of characters).	
modified	Specifies whether the host field is modified.	
numeric	Specifies whether the host field is limited to numeric input.	
penDetectable	Specifies whether the host field can be detected by a light-pen device.	
penSelectable	Specifies whether the host field can be selected by a light-pen device.	
protected	Specifies whether a user can add input in the host field. `True` indicates that you cannot add input.	
reverse	Specifies whether the host field is displayed in reverse video (switch foreground and background colors).	
startCol	Specifies the first column of the field.	
startRow	Specifies the first row of the field.	
text	Specifies the current text within the host field.	
underline	Specifies whether the host field is underlined.	

Properties of screen property verification points

You can create screen property verification points to test the screen in your application. When you record the verification point, a baseline of the screen is created. Every time you play back the script, the screen is compared to check if any changes have occurred. This helps in identifying any mismatch in screen.

Table 1 shows the properties, descriptions, and default values of screen property verification points.

Table 16. Screen property verification points properties

Property	Description	Valid values	Default value
.fieldCount	Specifies the number of fields on the screen.		
accessibilityEnabled	Enables the accessibility API in the terminal screen when set to `true`.	`true` or `false`	`true`
alignmentX	Specifies the component position within BoxLayout. If border layout is specified as X_Axis, you can change the alignmentY to -1 for top or 1 for bottom. If the Y_Axis is specified, you can change the alignmentX to -1 for left or 1 for right.		`0.5`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
alignmentY	Specifies the component position within BoxLayout. If border layout is specified as Y_Axis, you can change the alignmentX to -1 for top or 1 for bottom. If the X_Axis is specified, you can change the alignmentY to -1 for left or 1 for right.		`0.5`
autoFontSize	Automatically selects the best font size whenever the window is resized. `True` indicates that any calls to setFontSize() is ignored.	`true` or `false`	`true`
autoPack	Automatically packs the subcomponents of the screen property verification point when set to `true`.	`true` or `false`	`false`
autoscrolls	Specifies that when set to `true`, mouse-drag events are synthetically generated when the mouse is dragged outside of the component bounds and mouse motion has paused while the button continues to be held down.	`true` or `false`	`false`
background	Specifies the background color. You can chose from basic, system, or specify the RGB colors to replace the default colors.		`java.awt.Color[r=212,g=208,b=200]`
background3DColour	Specifies the background three-dimensional color.	`true` or `false`	`true`
backgroundSet	Specifies whether the background color is explicitly set for the component. If `false`, the component inherits its background color from an ancestor.	`true` or `false`	`true`
blockCursor	Displays the full height block cursor, or underscored cursor. If the window is currently in insert mode, the block or underscore cursor is not displayed until you exit insert mode. In insert mode, the window displays a half-height cursor.`True` causes the window to display a full height block cursor and `false` causes the window to display an underscored cursor.	`true` or `false`	`false`
bounds	Specifies the bounds of the rectangle of the object in screen coordinates. **Restriction:** When you play back the recorded test scripts by using Google Chrome on the Linux op-		`java.awt.Rectangle[x=0,y=0,width=740,height=`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
	erating system, the bounds property verification points of the document objects might capture incorrect values.		
centered	Causes the window to automatically center the text area and operator information area (OIA) within its current boundaries when set to true.	`true` or `false`	`true`
class	Specifies the Rational® Functional TesterHCL OneTest™ UI class name. For example, `HtmlTable` is the class name for a <Table> element.		`com.ibm-.eNetwork-.beans.HOD-.Screen`
codePage	Specifies the code page property.		`037`
columns	Specifies the number of columns. This value depends on the screen size chosen for the connection configuration.		`80`
component	Specifies an object with a graphical representation that can be displayed on the screen and that the user can interact with the component.		`Terminal`
componentCount	Specifies the number of components in the panel.		
cursorCol	Specifies the current column location of the cursor on the host screen.		
cursorRow	Specifies the current row location of the cursor on the host screen.		
cursorSet	Specifies whether the cursor for the component is explicitly set or inherited from an ancestor. When `true`, the cursor is set explicitly. When `false`, the component inherits its cursor from an ancestor.	`true` or `false`	`false`
cursorVisible	Specifies whether the cursor is made visible by screen.	`true` or `false`	`true`
dBCSInputVisible	Specifies the dBCSInputVisible property (3270 and 5250 DBCS sessions only). When `true`, the window displays the double-byte character set (DBCS) input field.	`true` or `false`	`false`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
debugGraphics-Option	Enables or disables the diagnostic information about every graphics operation that is performed within the component or one of its children.		`0`
displayable	Specifies whether a component can be displayed. The component can be displayed when it is connected to a native screen resource.	`true` or `false`	`true`
doubledBuffered	Specifies whether the receiving component uses a paint buffer. When set to `true`, the painting is performed to an off-screen buffer, and then copied to the window.	`true` or `false`	`true`
enabled	Specifies that the component responds to user input and generates events.	`true` or `false`	`true`
fixedFontSize	Specifies whether the font size is fixed.		
focus	Specifies whether the component has focus.	`true` or `false`	`false`
focusCycleRoot	Specifies whether the container is the root of a focus traversal cycle. When focus enters a traversal cycle, it cannot leave it by focus traversal unless, one of the up or down-cycle keys is pressed. Normal traversal is limited to this container, and all its descendantxs that are not descendants of inferior-focus cycle roots.	`true` or `false`	`false`
focusOwner	Specifies whether the component is the focus owner.	`true` or `false`	`false`
focusTraversable	Specifies whether the component can become the focus owner.	`true` or `false`	`true`
focusTraversal-KeysEnabled	Specifies whether focus traversal keys are enabled for the component. Components for which focus traversal keys are disabled receive key events for focus traversal keys. Components for which focus traversal keys are enabled do not process these events; instead, the events are automatically converted to traversal operations.	`true` or `false`	`true`
focusTraversal-PolicySet	Specifies whether the focus traversal policy has been explicitly set for the component. If this setting is `false`, the component inherits its focus traversal policy from an ancestor.	`true` or `false`	`false`
focusable	Specifies whether the component can have focus.	`true` or `false`	`true`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
font	Specifies the name, style, and size of the text font within the component.		`com,ratio-nal.test-.ft.val-ue/FontIn-fo[name=Mono-spaced,style=0,size=15]`
fontname	Specifies the font name. The name must denote a mono-space font, such as courier or monospaced.		`Monospaced`
fontSet	Specifies whether the font of the component is explicitly set or inherited from its ancestor. When `true`, the font is explicitly set. When `false`, the font is inherited from an an-cestor.	`true` or `false`	`false`
fontSize	Specifies the font size. This property is ignored when the autoFontSize property is set to `true`.		`15`
fontSizeBounded	Rejects any font or font size that can cause the screen text to exceed current screen boundaries when set to `true`.	`true` or `false`	`true`
fontStyle	Specifies the font style. The styles can be combined for mixed styles. The possible values are: **Value** / **Constant** 0 — java.awt.Font.PLAIN 1 — java.awt.Font.BOLD 2 — java.awt.Font.ITALIC	Integers from `0` though `2`	`0`
footerPlace	Specifies the footer placement of the print screen. This method is valid only on Java2.	`PRT_SCRN_PLACE_LEFT`, `PRT_SCRN_PLACE_CENTER`, or `PRT_SCRN_PLACE_-RIGHT`	
footerText	Specifies the footer text of the print screen. This method is valid only on Java2.		
foreground	Specifies the foreground color. You can chose from basic, system, or specify RGB colors to replace the default col-ors.		`ja-va,awt,Col-or[r=0,g=0,b=0]`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
foregroundSet	Specifies whether the foreground color of the component is explicitly set or inherited from its ancestor. When `true`, the foreground color is explicitly set. When `false`, the foreground color is inherited from an ancestor.	`true` or `false`	`true`
function	Specifies the function of the component.		`Host On-Demand`
headerPlace	Specifies the header placement of the print screen. This method is valid only on Java2.		
headerText	Specifies the header text of the print screen. This method is valid only on Java2.		
height	Specifies the height of the component.		`570`
hiddenFieldDisplay	Specifies whether to display hidden fields.	`true` or `false`	`false`
hiddenFieldDisplayed	Specifies whether the hidden fields are displayed.	`true` or `false`	`false`
hideUnprotectedURLsMode	Specifies whether URLs in unprotected fields are rendered as hotspots.	`true` or `false`	`true`
iMEAutoStart	Specifies whether IMEAutoStart is enabled.	`true` or `false`	`false`
ignoreRepaint	Specifies whether the component ignores all repaint events.	`true` or `false`	`false`
lightPenMode	Enables light-pen support. Valid for 3270 and CICS® sessions only.	`true` or `false`	`false`
lightweight	Specifies that a component does not have a native toolkit peer. Subclasses of components and containers, other than the ones defined in this package, such as button or scrollbar are lightweight. All the swing components are lightweights.	`true` or `false`	`true`
location	Specifies the location of the upper-left corner of the component.		`java.awt-.Point[x=0,y=0]`
locationOnScreen	Specifies the location of a component in the form of a point specifying the component upper-left corner in the coordinate space of the screen.		`java.awt-.Point[x=5,y=85]`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
managingFocus	Specifies whether the component focus traversal keys are Ctrl+Tab and Ctrl+Shift+Tab.	`true` or `false`	`false`
markedArea-PrintingEnabled	Enables printing only a marked area of the screen when set to `true`.	`true` or `false`	`true`
maximumSize	Specifies the maximum size for the component.		`java.awt-.Dimen-sion[width=2147483647,height=`
maximumSize-Set	Specifies whether the maximum size is set.	`true` or `false`	`false`
minimumSize	Specifies the minimum size for the component.		`java.awt-.Dimen-sion[width=720,height=531]`
minimumSizeSet	Specifies whether the minimum size is set.	`true` or `false`	`false`
morePasteData-Available	Specifies whether more data to paste is available.	`true` or `false`	`false`
mouseEnabled	Specifies whether mouse events are handled by the screen. When **true** the topmost component will intercept and handle all the mouse events. When **false**, the low-er-level components will intercept and handle the mouse events.	`true` or `false`	`true`
name	Value of the name attribute (form elements and frames only).		
oIAVisable	Specifies that the screen displays the operator informa-tion area (OIA) when set to true.	`true` or `false`	`true`
opaque	Specifies whether the component is set to opaque. If so, the painting system does not paint anything behind the component.	`true` or `false`	`true`
optomized-DrawingEnabled	Specifies whether optimized drawing is enabled.	`true` or `false`	`true`
paintingTile	Specifies whether the component is currently painting a tile. When `true`, paint is called again for another tile. When `fasle`, the tile is not being painted or the last tile is painted.	`true` or `false`	`true`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
preferredSize	Specifies the best size for the component. Certain layout managers ignore this property.		`java.awt-.Dimen-sion[width=720,height=531]`
preferredSizeSet	Specifies that the preferred size is set to a non-null value when `true`.	`true` or `false`	`false`
requestFocusEn-abled	Specifies that the component gets the keyboard focus.	`true` or `false`	`true`
rows	Represents the value of the rows attribute of a TEXTAREA element, indicating the size of the edit control in number of rows of text.		
rule	Displays rule lines when set to true.	`true` or `false`	`false`
sessionType	Specifies the session type.	Integers from `1` through `5`	`1`

Constant	Value	Session type
SESSION_TYPE_-3270_STR	1	3270 (default)
SESSION_TYPE_-5250_STR	2	5250
SESSION_TYPE_-CICS_STR	4	CICS®
SESSION_TYPE_-3270_PRT_STR	5	3270 printer

setRasterFont	Specifies the raster font.		
showURLsMode	Specifies whether the URLs are displayed as hotspots, and if so, whether they are rendered with underlines or as buttons.		`underlined-URLs`
showing	Specifies whether the component is showing on screen. This means that the component must be visible and must be in a container that is visible.		`true`
size	Value of the size attribute of an element. For a select element, this indicates the number of items displayed at the same time in the list. If size>1, the list appears as a		`java.Dimen-sion[width=740,height=531]`

Table 16. Screen property verification points properties (continued)

Property	Description	Valid values	Default value
	list box, otherwise the list appears as a combination drop-down box.		
skipPrintingDialog	Specifies whether the print dialog is suppressed in printing screen.	`true` or `false`	`false`
toolTipText	Specifies the text that you want in fly-over text or hover help.		
traceLevel	Specifies the traceLevel property (java.lang.Integer) value.		`0`
traceName	Specifies the trace name for this object.		`Terminal`
uIClassID	Specifies the name of the L&F class that renders this component.		`PanelUI`
valid	Specifies whether the component is valid. A component is valid when it is correctly sized, positioned within its parent container, and all its children are valid.	`true` or `false`	`false`
validateRoot	Specifies that the entire tree beginning with this root will be validated.	`true` or `false`	`false`
verifyInputWhenFocusTarget	Specifies whether the input verifier for the current focus owner is called before this component requests focus.	`true` or `false`	`true`
version	Specifies the version.		
visible	Specifies that the component is visible when set to `true`.	`true` or `false`	`true`
visibleRect	Specifies the component as a visible rectangle. The intersection of this component's visible rectangle and all of its ancestors' visible rectangles.		`java.awt-.Rectan-gle[x=0,y=0,width=740,height=`
width	Specifies the width of the component.		`740`
x	Specifies the current x coordinate of the component origin.		`0`
y	Specifies the current y coordinate of the component origin.		`0`

Properties of display property verification points

You can create display property verification points to test the display in your application. When you record the verification point, a baseline of the display is created. Every time you play back the script, the display is compared to check if any changes have occurred. This helps in identifying any mismatch in display.

Table 1 shows the properties, descriptions, and default values of display property verification points.

Table 17. Display property verification point properties

Property	Description	Valid values	Default value
accessibilityEnabled	Enables the accessibility API in the terminal screen when set to true.	`true` or `false`	`true`
alignmentX	Specifies the component position within BoxLayout. If border layout is specified as X_Axis, you can change the alignmentY to -1 for top or 1 for bottom. If the Y_Axis is specified, you can change the alignmentX to -1 for left or 1 for right.		`0.5`
alignmentY	Specifies the component position within BoxLayout. If border layout is specified as Y_Axis, you can change the alignmentX to -1 for top or 1 for bottom. If the X_Axis is specified, you can change the alignmentY to -1 for left or 1 for right.		`0.5`
allocateSpaceForLamAlef	Specifies whether LamAlef is expanded or compressed. This property applies to Arabic sessions only.		`LAMALEFOFF`
autoConnect	Specifies whether to automatically connect to the host when the host property is set.	`true` or `false`	`true`
autoFontSize	Automatically selects the best font size whenever the window is resized. Calls to setFontSize() are ignored when set to `true`.	`true` or `false`	`true`
autoPack	Automatically packs its subcomponents when set to `true`.	`true` or `false`	`false`
autoReconnect	Specifies whether to automatically reconnect to the host after the host connection is disconnected.	`true` or `false`	`true`
autoscrolls	Specifies that when set to `true`, mouse-drag events are synthetically generated when the mouse is dragged outside of the bounds of the component and mouse movement is paused while the button continues to be held down.	`true` or `false`	`false`
bIDIMode	Specifies whether to enable or disable BIDI functions, such as character shaping. This property applies to Arabic VT sessions only.	`BIDIMODEON` and `BIDIMODEOFF`	`BIDIMODEON`
background	Specifies the background color. You can chose from basic, system, or specify RGB colors to replace the default colors.		`java.awt.Color[r=212,g=208,b=200]`
background3DColour	Specifies the background three-dimensional color.	`true` or `false`	`true`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
backgroundSet	Specifies whether the background color of the component is explicitly set or inherited from an ancestor. If `true`, the component background color is explicitly set. If `false`, the component inherits background color from an ancestor.	`true` or `false`	`true`
blockCursor	Displays the full height block cursor, or underscored cursor. If the window is currently in insert mode, the block or underscore cursor is not displayed until you exit insert mode. In insert mode, the window displays a half-height cursor. `True` causes the window to display a full height block cursor. `False` causes the window to display an underscored cursor.	`true` or `false`	`false`
bounds	Specifies the bounds of the rectangle of the object in window coordinates.		`java.awt-.Rectan-gle[x=0,y=0,width=740,height=`
cICSGWCode-Page	Specifies the CICS® gateway.		`000`
cICSServerName	Specifies the CICS server name.		
centered	Automatically centers the text area and OIA within its current boundaries when set to true.	`true` or `false`	`true`
class	Specifies the Rational® Functional TesterHCL OneTest™ UI class name. For example, `HtmlTable` is the class name for a <Table> element.		`com.ibm-.eNetwork-.beans.HOD-.Terminal`
codePage	Specifies the codePage property.		`037`
columns	Specifies the number of columns.		`80`
commStarted	Specifies whether the connection with the host is started. A return value of `true` indicates that the connection has been started, but the connection might not be ready for interaction.		`true`
commStatus	Specifies the status of communications with the host. The possible values are as follows: **Constant** / **Value** / **Description** CONNECTION_INIT / 1 / Initial state (no connection with host).	Integers from `1` through `6`	`5`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
	Constant / **Value** / **Description**: CONNECTION_PND_INACTIVE / 2 / Stop communications in progress. CONNECTION_INACTIVE / 3 / The connection is stopped. CONNECTION_PND_ACTIVE / 4 / Start communications in progress. CONNECTION_ACTIVE / 5 / Connection is established. CONNECTION_READY / 6 / Negotiations started.		
component	Specifies an object with a graphical representation that can be displayed on the window, and that the user can interact with the component.		`Terminal`
componentCount	Specifies the number of components in the panel.		`1`
connectionTimeout	Specifies the connection timeout value.		`0`
copyAltSignLocation	Specifies the mode of cut or copy for the sign of a number. If `true`, the sign character is put in front of the number. If `false`, the sign character is in the same location relative to the number as it is displayed on the screen.	`true` or `false`	`false`
copyOnlyIfTrimmed	Specifies whether to set the copy error when there is no trim. . If `true`, a copy error is set when there is no trim. If `false`, the entire screen is copied when there is no trim.	`true` or `false`	
cursorDirection	Determines whether the cursor direction is left-to-right or right-to-left. This property applies to BIDI visual VT sessions only.	`CURSOR_LEFTTORIGHT` or `CURSOR_RIGHTTOLEFT`	`CURSOR_LEFTTORIGHT`
cursorMovementState	Specifies whether users can move the cursor with a mouse click within the presentation space. This property currently applies to VT sessions only.	`true` or `false`	`true`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
cursorSet	Specifies whether the cursor of the component is explicitly set or inherited from an ancestor. If `true`, the cursor is explicitly set. If `false`, the component inherits its cursor from an ancestor.	`true` or `false`	`false`
cursorVisible	Specifies whether the cursor is made visible by screen.	`true` or `false`	`true`
dBCSInputVisible	Specifies the dBCSInputVisible property (3270 and 5250 DBCS sessions only). When `true`, the screen displays the DBCS input field.	`true` or `false`	`false`
debugGraphics-Option	Specifies whether the diagnostic information about every graphics operation that is performed within the component or one of its children is enabled or disabled.		`0`
deviceName	Specifies the device name.		
deviceName-Ready	Specifies whether the device name is ready. This method is only valid for 3270 sessions. If `true`, the device name is ready. If `false`, the device name is not ready.	`true` or `false`	`true`
displayable	Specifies whether the component can be is displayed. A component can be displayed when it is connected to a native screen resource.	`true` or `false`	`true`
doubledBuffered	Specifies whether the receiving component uses a buffer to paint. If set to true, the painting is performed to an offscreen buffer and then copied to the screen.	`true` or `false`	`true`
eNPTUI	Indicates whether to enable the Enhanced Non-Programmable Terminal User Interface (ENPTUI) functionality. This property can be enabled for 5250 sessions only.	`true` or `false`	`false`
enabled	Specifies that the component responds to user input and generates events.	`true` or `false`	`true`
entryAssist_bell	Enables or disables the audible signal when the cursor enters the column set for the End of Line Signal column. The value of `true` turns on the signal and `false` turns off the signal.	`true` or `false`	`false`
entryAssist_bell-Col	Controls the column number at which you want the audible signal for the End of Line to sound. The audible signal sounds only if the EntryAssist_bell property is set to `true`.	Valid column numbers	`75`
entryAssist_DOC-mode	Enables or disables the entry assist features. The entry assist (DOC mode) features make it easier to edit text documents in a	`true` or `false`	

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
	3270 Display session. The value of `true` turns DOC mode on and `false` turns DOC mode off.		
entryAssist_DOC-wordWrap	This setting enables or disables word wrap. When word wrap is enabled, a word that is typed at the right margin is moved entirely to the first unprotected field in the next row, provided that the unprotected field has enough blank space to the left to contain the word. The area on the previous row vacated by the word is filled with spaces. If the unprotected field does not have enough blank space at the left to contain the word, then the word is not moved. The effect is the same as if word wrap were not enabled. The value of `true` turns word wrap on and `false` turns word wrap off.	`true` or `false`	`true`
entryAssist_end-Col	Controls the right margin for DOC mode. When the DOC mode is on, the right-most cursor position in a row is the last unprotected character position to the left of the last column.	Valid column numbers	`80`
entryAssist_s-tartCol	Controls the left margin for DOC mode. When DOC mode is on, the left-most cursor position in a row is the first unprotected character position to the right of the start column.	Valid column numbers	`1`
entryAssist_tab-stop	Controls the number of spaces that is skipped when the Tab key is pressed.	Valid numbers of spaces	`8`
entryAssist_tab-stops	Controls the columns at which you want tab stops. When tab stops are set, pressing the Tab key causes the cursor to skip to one of the following, in order of occurance: • The next Tab stop in the same unprotected field on the same row. (Tab stops cannot be defined outside the left or right margin.) • The first character position in the next unprotected field on the same row, if that character position is within the margins. • The first character position in the next unprotected field in a subsequent row, if that character position is within the margins. **Note:** Characters in an unprotected field that are skipped as the result of pressing the Tab key are not set to blanks.	String representations of arrays of columns to use as tab stops. For example: `5`, `10`, `15`, `20`, `25`	

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
	Only nulls that the cursor skips as the result of pressing Tab key are set to blanks.		
focus	Specifies whether the component has focus.	`true` or `false`	`false`
focusCycleRoot	Specifies whether the container is the root of a focus traversal cycle. Once focus enters a traversal cycle, typically it cannot leave it via focus traversal unless one of the up or down-cycle keys is pressed. Normal traversal is limited to this container, and all of the descendants of the container that are not descendants of inferior focus cycle roots.	`true` or `false`	`false`
focusOwner	Specifies whether the component is the focus owner.	`true` or `false`	`false`
focusTraversable	Specifies whether the component can become the focus owner.	`true` or `false`	`true`
focusTraversal-KeysEnabled	Specifies whether focus traversal keys are enabled for the component. Components for which focus traversal keys are disabled receive key events for focus traversal keys. Components for which focus traversal keys are enabled do not process these events: instead the events are automatically converted to traversal operations.	`true` or `false`	`true`
focusTraversal-PolicySet	Specifies whether the focus traversal policy of the component is explicitly set or inherited from its ancestor. If `true`, the focus traversal policy is set explicitly. If `false`, the component inherits its focus traversal policy from an ancestor.	`true` or `false`	`false`
focusable	Specifies whether the component can have focus.	`true` or `false`	`true`
font	Specifies the name, style, and size of the text font within the component.		`com,ratio-nal.test-.ft.val-ue/FontIn-fo[name=Mono-spaced,style=0,size=15]`
fontName	Specifies the font name. The name must denote a monospace font, such as courier or monospaced.		`Monospaced`
fontSet	Specifies whether the font of the component is explicitly set or inherited from an ancestor. If `true`, the font is explicitly set. If `false`, the font is inherited from an ancestor.	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
fontSize	Specifies the font size. This property is ignored when the autoFontSize property is set to `true`.		`15`
fontSizeBounded	Causes the screen to reject any font or font size that would cause the screen text to exceed current screen boundaries when set to `true`.	`true` or `false`	`true`
fontStyle	Specifies the font style. The styles can be combined for mixed styles. The possible values are as follows: **Value** / **Constant** 0 — java.awt.Font.PLAIN 1 — java.awt.Font.BOLD 2 — java.awt.Font.ITALIC	Integers from `0` though `2`	`0`
footerPlace	Specifies the footer placement of the print screen. This method is valid only on Java2.	`PRT_SCRN_PLACE_LEFT`, `PRT_SCRN_PLACE_CENTER`, or `PRT_SCRN_PLACE_RIGHT`	
footerText	Specifies the footer text of the print screen. This method is valid only on Java2.		
foreground	Specifies the foreground color. You can chose from basic, system, or specify RGB colors to replace the default colors.		`java,awt,Color[r=0,g=0,b=0]`
foregroundSet	Specifies whether the foreground color of the component is explicitly set or inherited from its ancestor. If `true`, the foreground color is set explicitly. If `false`, the foreground color is inherited from an ancestor.	`true` or `false`	`true`
function	Specifies the function of the component.		`Host On-Demand`
graphicsCellSize	Specifies the graphic cell size.		`0`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
headerPlace	Specifies the header placement of the print screen. This method is valid only on Java2.		
headerText	Specifies the header text of the print screen. This method is valid only on Java2.		
height	Specifies the height of the component.		`570`
hiddenFieldDisplay	Specifies whether to display hidden fields.	`true` or `false`	`false`
hiddenFieldDisplayed	Specifies whether hidden fields are displayed.	`true` or `false`	`false`
hideUnprotectedURLsMode	Specifies whether the URLs in unprotected fields are rendered as hot spots.	`true` or `false`	`true`
history	Specifies whether the history log is maintained. If `true`, the history log is maintained. If `false`, the history log is not maintained.	`true` or `false`	`true`
historySize	Specifies the size of the internal planes that are used to store history log information.		`64`
host	Specifies the name of the host that is associated with the session bean. Communication between the session bean and the host is started after a call to startCommunication.		
hostBackup1	Specifies the host name or IP address of the backup1 server. Displayed as `Destination address` of backup1 on property panels. Applies to all session types.		
hostBackup2	Specifies the host name or IP address of the backup2 server. Displayed as `Destination address` of backup2 on property panels. Applies to all session types.		
hostGraphics	Specifies whether to enable the host graphics functionality. This property can be enabled for 3270 sessions only.	`true` or `false`	`false`
iMEAutoStart	Specifies whether IMEAutoStart is enabled.	`true` or `false`	`false`
ignoreRepaint	Specifies whether the component ignores all repaint events.	`true` or `false`	`false`
ignoreWellKnownTrustedCAs	Specifies whether the component ignores signer certificates. This property applies to SSL sessions only. If `true`, the component ignores WellKnownTrustedCAs.class signer certificates. If `false`, the component uses WellKnownTrustedCAs.class signer certificates.	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
insertOffOnAIDKEY	Sets the InsertOffOnAIDKEY property of session. This property is valid for 3270 and CICS sessions only. **Insert mode is set as follows** / **Typing any `AIDKEY` performs as follows** `On` and isInsertOffOnAIDKEY is `true` / Turns insert mode `off` `On` and isInsertOffOnAIDKEY is `false` / Has no effect on the insert mode `On` / Does not turn insert mode `on` regardless of the state of isInsertOffOnAIDKEY	`true` or `false`	`false`
keyStoreFilePath	Specifies the path and name of the keystore file that is on the client workstation containing the client public and private keys.	Valid path and file name of the keystore file	
keyStorePassword	The password that is required to open the keystore file that is on the client workstation.	Correct password to open the keystore file	no password
lUMLicensing	Specifies the license method.	`LUM` or `HOD`	`HOD`
lUMPort	Specifies the LUM port.	Valid port numbers	`80`
lUMServer	Specifies the LUM server name.	Valid LUM server names	
lUName	Specifies the LU name that is used during enhanced negotiation. Maximum length of LU name is 17 characters. This property is valid only when the tNEnhanced property is true. This method is valid for 3270 sessions only. **Note:** For best results, first call the isValidLUName(String luName) function first to check the validity of the string.		
lUNameBackup1	The name of the LU or LU Pool, defined at the backup1 server, to which you want the session to connect to. This is displayed as `LU or Pool Name` of backup1 on property panels. Applies to 3270 Display and 3270 Printer session types.	Valid LU or LU pool	

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
lUNameBackup2	The name of the LU or LU Pool, defined at the backup2 server, to which you want the session to connect to. This is displayed as `LU or Pool Name` of backup2 on property panels. Applies to 3270 Display and 3270 Printer session types.	Valid LU or LU pool names	
lastHostWithoutTimeout	Indicates whether the connection to the last configured host occurred without timeout.	`true` or `false`	`true`
lightPenMode	Causes the screen to enable light pen support. Valid for 3270 and CICS sessions only.	`true` or `false`	`false`
lightweight	Specifies that a component does not have a native toolkit peer. Subclasses of component and container, other than the ones defined in this package, such as button or scrollbar are lightweight. All the swing components are lightweights.	`true` or `false`	`true`
location	Specifies the location of the upper-left corner of the component.		`java.awt-.Point[x=0,y=0]`
locationOnScreen	Specifies the location of the component in the form of a point specifying the component's upper-left corner in the coordinate space of screen.		`java.awt-.Point[x=5,y=85]`
managingFocus	Specifies whether the component focus traversal keys are Ctrl +Tab and Ctrl+Shift+Tab.	`true` or `false`	`false`
markedAreaPrintingEnabled	Enables printing only a marked area of the screen when set to `true`.	`true` or `false`	`true`
maximumSize	Specifies the maximum size for the component.		`java.awt-.Dimen-sion[width=2147483647,height=`
maximumSizeSet	Specifies whether the maximum size is set.	`true` or `false`	`false`
minimumSize	Specifies the minimum size for the component.		`java.awt-.Dimen-sion[width=720,height=531]`
minimumSizeSet	Specifies whether the minimum size is set.	`true` or `false`	`false`
morePasteDataAvailable	Specifies whether more data to paste is available.	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
mouseEnabled	Specifies whether mouse events are handled by screen. When `true` the topmost component will intercept and handle all the mouse events. When `false`, the lower-level components will intercept and handle the mouse events.	`true` or `false`	`true`
name	Specifies the value of the name attribute (form elements and frames only).		
negotiateCResolution	Specifies whether to enable negotiate contention resolution.	`true` or `false`	`true`
numeralShape	Specifies the numeral shape as nominal, national or contextual for strings sent to the presentation space. This applies to Arabic Hosts only.	`NOMINAL`, `NATIONAL` or `CONTEXTUAL`	`NOMINAL`
numeralShapeDisp	Determines how numerals are shaped. This property applies to Arabic VT sessions only.	`NOMINAL_DISP`, `NATIONAL_DISP` or `CONTEXTUAL_DISP`	`CONTEXTUAL_DISP`
numericFieldLock	Specifies whether to limit the field characters of a session to numeric values. When set to `true`, users can type only characters 0 through 9, -, +, period (.), and comma(,) in fields that are defined by a host application as numeric. This property is valid for 3270 and CICS sessions only.	`true` or `false`	`false`
numericSwapEnabled	Enables numeric swapping. This property applies to Arabic 3270 sessions only.	`true` or `false`	`true`
oIAVisable	Specifies that the window displays the operator information area (OIA) when set to true.	`true` or `false`	`true`
opaque	Specifies whether the component is set to opaque. If so, the painting system does not paint anything behind the component.	`true` or `false`	`true`
optomizedDrawingEnabled	Specifies whether optimized drawing is enabled.	`true` or `false`	`true`
paintingTile	Specifies whether the component is currently painting a tile. If `true`, paint will be called again for another tile. If `false`, the tile is not being painted or the last tile is painted.	`true` or `false`	`true`
pasteFieldWrap	Enables wrap on the field. This property does not apply to VT sessions. If `true`, set wrap on field. If `false`, set normal wrap.	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
pasteLineWrap	Enables line wrap on field. If `true`, set line wrap on field. If `false`, set normal wrap.	`true` or `false`	`false`
pasteStopAtProtectedLine	Specifies whether a user can allow paste in a protected area. This property does not apply to VT sessions. If `true`, users cannot paste in a protected line. If `false`, users can paste normally	`true` or `false`	`false`
pasteTabColumns	Specifies the pasteTabColumns to set the number of columns represented by a tab. If this option is active, when a user presses the Tab key, the input skips to the column that is a multiple of this setting.	Size of tab in columns	`4`
pasteTabOptions	Specifies the pasteTabOptions.		`2`
pasteTabSpaces	Sets the pasteTabSpaces to set the number of spaces represented by a tab. If this option is active, when a user presses the Tab key, the input skips the number of spaces specified in this setting.	Spaces to advance for a tab	`1`
pasteToTrimmedArea	Specifies whether users can paste in trimmed areas. This property does not apply to VT sessions. If `true` users can paste into trimmed areas, if defined. If `false`, users can paste normally.	`true` or `false`	`false`
pasteWordBreak	Specifies whether the paste splits words. This property does not apply to VT sessions. If `true`, pasted words are not split. If `false`, words are pasted normally.	`true` or `false`	`true`
port	Specifies the port number on which the server is configured.		`23`
portBackup1	Specifies the port number on which the backup1 server is configured. Displayed as `Destination port` of backup1 on property panels. Applies to all session types.		`23`
portBackup2	Specifies the port number on which the backup2 server is configured. Displayed as `Destination port` of backup2 on property panels. Applies to all session types.		`23`
preferredSize	Specifies the best size for the component. Certain layout managers ignore this property.		`java.awt-.Dimen-sion[width=720,height=531]`
preferredSizeSet	Specifies that the preferred size is set to a non-null value when true.		`false`
printDestination	Specifies whether the output goes to a printer or to a file. If `true`, output goes to a printer. If `false`, output goes to a file.	`true` or `false`	`true`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
printerName	Specifies the name of the destination printer device.	Valid print destination printers	`LPT1`
printFileName	Specifies the name to be assigned to the print file.	Valid print file names	
proxyAuthenMethod	Specifies the authentication method between the Host On-Demand session and proxy server. Select one of the following: • Basic (HTTP only): The Host On-Demand session provides a user ID and password to the HTTP proxy server. • Clear Text (SOCKS v5 only): The Host On-Demand session provides an unencrypted user ID and password to the socks proxy server. • None: The Host On-Demand session does not provide a user ID and password to the HTTP proxy or socks server.  **Note:** If you select `basic` or `clear text` as the proxy authentication method, you must specify a User ID and Password.		`SESSION_PROXY_AUTHEN_NONE`
proxyServerName	Specifies the host name or IP address of the HTTP or socks proxy server.		
proxyServerPort	Specifies the TCP port number of the HTTP or socks proxy server.		`1080`
proxyType	Specify the type of proxy server that a host session uses. • Default Browser Setting • HTTP Proxy • SOCKS v4 • SOCKS v5 • SOCKS v4, if v5 unavailable		`SESSION_PROXY_BROWSER_DEFAULT`
proxyUserID	Specifies the user ID that the Host On-Demand session provides to authenticate with the HTTP or socks proxy server.		
proxyUserPassword	Specifies the password that the Host On-Demand session provides to authenticate with the HTTP or socks proxy server.		

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
roundTrip	Specifies whether the roundTrip is set to `ON` or `OFF`. This method applies to bidirectional hosts only.	`ON` or `OFF`	`ON`
rTLUnicodeOverride	Enables or disables the RTL unicode override option. This applies to BIDI 5250 Hosts only.	`RTLUNICODEON` or `RTLUNICODEOFF`	`RTLUNICODEOFF`
requestFocusEnabled	Specifies that the component gets the keyboard focus.	`true` or `false`	`true`
rows	Specifies the value of the rows attribute of a TEXTAREA element, indicating the size of the edit control in number of rows of text.		`24`
rule	Displays rule lines when set to `true`.	`true` or `false`	`false`
sLPAS400Name	Connects a session to a specific iSeries™ server. Displayed as AS/400® Name (SLP) on property panels. Applies to 5250 Display and 5250 Printer session types. Use the fully-qualified SNA CP name, for example, USIBMNM.RAS400B.		
sLPEnabled	Specifies whether the service-location protocol is used. If `true`, use Service Location Protocol (SLP). If `false`, do not use SLP	`true` or `false`	`false`
sLPMaxWaitTime	Specifies the SLPMaxWaitTime in milliseconds to wait for service response. This property is valid when the SLPEnabled property is `true` only.		`200`
sLPScope	Sets the service location protocol (SLP) scope. Displayed as `Scope` under `SLP Options` on property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, and 5250 Printer session types.		
sLPThisScopeOnly	Specifies whether the session is established only with a server that supports the provided scope. This property is valid only when the SLPEnabled property is `true` and there is a SLPScope provided.	`true` or `false`	`false`
sSHPublicKeyAlias	Specifies the SSHPublicKeyAlias.		`mykey`
sSHPublicKeyAliasPassword	Specifies the SSHPublicKeyAliasPassword.		
sSL	Specifies whether to use the Secure Socket Layer (SSL) feature. If `true`, enable SSL. If `false`, disable SSL	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
sSLBrowser-KeyringAdded	Specifies the SSLBrowserKeyringAdded property of session. If `true`, add the session to the HOD client keyring. If `false`, do not add the session to HOD client keyring.	`true` or `false`	`false`
sSLCertificate-Hash	Specifies the SSLCertificateHash.		
sSLCertificate-Name	Specifies the SSLCertificateName.		
sSLCertificate-Password	Specifies the SSLCertificatePassword.		
sSLCertificate-PromptBefore-Connect	Specifies whether the client is prompted before connecting to the server. If `true`, prompt the client. If `false`, do not prompt the client.	`true` or `false`	`false`
sSLCertificate-PromptHow-Often	Specifies how often the client is prompted.	`SESSION_-SSL_CERTIFI-CATE_PROMP-T_EACH_CON-NECT`, `SESSION_-SSL_CERTIFI-CATE_PROMPT_-FIRST_CONNECT` or `SESSION_-SSL_CERTIFI-CATE_PROMPT_-ONLY_ONCE`.	`SESSION_-SSL_-CERTIFI-CATE_PROMP-T_FIRST_-CONNECT`
sSLCertificate-Provided	Specifies whether the client has a certificate. If `true`, the client has a certificate. If `false`, the client does not have a certificate.	`true` or `false`	`false`
sSLCertificateRe-membered	Specifies the SSLCertificateRemembered property of session. If `true`, sets SSLCertificatePromptHowOften to FIRST_CONNECT. If `false`, sets SSLCertificatePromptHowOften to EACH_CONNECT.	`true` or `false`	`true`
sSLCertificate-Source	The certificate can be kept in the client browser or a dedicated security device, such as a smart card, or local or network-accessed file. This property is displayed as `Certificate Source` on property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, 5250 Printer, and VT Display session types.	`SSL_CERTIFI-CATE_IN_CSP` for a certificate in a browser or security device or `SSL_CERTIFI-`	`SESSION_-SSL_-CERTIFI-CATE_IN_URL`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
		`CATE_IN_URL` for a certificate in a URL or file	
sSLCertificate-URL	Specifies the default location of the client certificate. The location is displayed as a URL or a path and filename in property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, 5250 Printer, and VT Display session types. The URL protocols depend on the capabilities of your browser. Most browsers support HTTP, HTTPS, FTP, and FTPS.		
sSLServer-Authentication	Specifies whether the SSL server authentication is enabled.	`true` or `false`	`false`
sSLTelnetNegoti-ated	Specifies whether the SSL is negotiated on a Telnet connection. Set this property to `true` only if you are connecting to a Telnet server that supports IETF Internet-Draft `TLS-based Telnet Security`. This Internet-Draft defines the protocol for doing the SSL Handshake over a Telnet connection. Set the SSL property to `true` also.	`true` or `false`	`false`
sSOCMServer	Specifies the sso_cmserver property. Valid values are the address strings of back-end servers and applications that respond to SSO queries.		
sSOEnabled	Specifies that the session is SSO enabled (`true`).	`true` or `false`	`false`
sSOUseKer-berosPassticket	Specifies whether the SSO layer uses the client-side Kerberos support to acquire a Kerberos passticket for login. If `true`, this property instructs the SSO layer to use client-side Kerberos support. If `false`, this property instructs the SSO layer to not use client-side Kerberos support.	`true` or `false`	`false`
sSOUseLocalI-dentity	Specifies whether the SSO layer uses the local OS userID in the SSO process. If `true`, this property instructs the client to use the local OS user ID in SSO process. If `false`, this property instructs the client not to use the local OS user ID in SSO process.	`true` or `false`	`false`
screenSize	Specifies the screen size.		`2`
securityProtocol	Specifies whether to use Transport Layer Security (TLS) v1.0 protocol, SSL protocol, or SSH protocol for providing security. If set to `TLS` (default), and if the server is TLS-enabled, then a TLS v1.0 connection is provided. If server is not TLS-enabled, then the server negotiates the connection down to SSL protocol.	`SESSION_PRO-TOCOL_TLS`, `SESSION_PRO-TOCOL_SSL`, or	`SESSION_-PROTOCOL_-TLS`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
	The value can be one of the following: Constant / Value / Description SESSION_PROTOCOL_TLS / `TLS` / Use TLS v1.0 protocol (default) SESSION_PROTOCOL_SSL / `SSL only` / Use SSL v3.0 protocol to provide security SESSION_PROTOCOL_SSH / `SSH` / Use SSH protocol v2.0	`SESSION_PROTOCOL_SSH`	
separateFiles	Specifies whether the print files are stored in separate files. This property applies only to HOD VT sessions, 3270 printer and 5250 printer sessions. Specify `true` to save the print jobs in separate files.	`true` or `false`	`false`
serviceMgrHost	Specifies the name for the HOD server.		
sessionID	Specifies the short name that you want to assign to this session (appears in the OIA). It must be unique to this configuration. This property is displayed as `Session ID` on property panels. It applies to all session types. This is not used when the proxyType is set to `BROWSER_DEFAULT`.		
sessionName	Specifies the name of the session.		
sessionType	Specifies the session type. The value can be one of the following integers: Constant / Value / Session Type SESSION_TYPE_3270_STR / `1` / 3270 (default) SESSION_TYPE_5250_STR / `2` / 5250 SESSION_TYPE_CICS_STR / `4` / CICS SESSION_TYPE_3270_PRT_STR / `5` / 3270 Printer	Integers from `1` through `5`	`1`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
showTextAttributesEnabled	Specifies the property to show text attributes. This property applies to logical BIDI VT sessions only.	`true` or `false`	`true`
showURLsMode	Specifies whether the URLs are displayed as hotspots. If so, it also specifies whether they are rendered in underlined text or buttons.		`underlined-URLs`
showing	Specifies whether the component is showing on screen. The component must be visible, and it must be in a container that is visible.	`true` or `false`	`true`
size	Value of the size attribute of an element. For a select element, this indicates the number of items that are displayed at once in the list. If size>1, it is displayed as a list, otherwise the list is displayed as a combination drop-down box.		`java.Dimension[width=740,height=531]`
skipPrintDialog	Specifies whether the print dialog box is suppressed in the printing screen.	`true` or `false`	`false`
smartOrdering	Specifies whether the segment of characters with different text attributes is ordered separately.		`SMART_ORDERING_OFF`
socksV4UserID	Specifies the user ID for use with SOCKS v4 connections.		
symmetricSwapEnabled	Specifies whether symmetric swapping is enabled (`true`). This property applies to Arabic 3270 sessions only.	`true` or `false`	`true`
tNEnhanced	Specifies that the enhanced session (TN3270E) parameters is negotiated when set to `true`.	`true` or `false`	`false`
textOrientation	Specifies whether the test orientation is left-to-right or right-to-left. This property applies to bidirectional sessions only.	`LEFTTORIGHT` or `RIGHTTOLEFT`	`LEFTTORIGHT`
textType	Specifies whether the textType is visual or logical. This property applies to bidirectional sessions only.	`VISUAL` or `LOGICAL`	`VISUAL`
textTypeDisp	Determines whether the session works in logical or visual mode. This property applies to BIDI VT sessions only.	`LOGICAL_DISP` or `VISUAL_DISP`	`LOGICAL_DISP`
thaiDisplayMode	This method applies to Thai host machines only. The possible values are as follows: **Value** — **Mode** 1 — Non-composed mode. 2 — Composed mode. 3 — Composed mode with space alignment.	Integers between `1` through `5`	`5`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
	Value **Mode**		
	4 Composed mode with EOF alignment.		
	5 Composed mode with space and EOF alignment.		
timeout	Specifies the amount of time in milliseconds that the system waits for data. If no data is received for the specified amount of time, the session is disconnected. A value of 0 specifies that system will not timeout.		`0`
timeoutNoData-Initialization	Specifies whether to time out if no data is received at session initialization.	`true` or `false`	`false`
toolTipText	Specifies fly-over or hover help text.		
traceLevel	Specifies the traceLevel property (java.lang.Integer) value.		`0`
traceName	Specifies the trace name for the object.		`Terminal`
trimRectRemain-AfterEdit	Specifies whether the trim rec remains after cutting, copying, or pasting. If `true`, trim rec remains after cutting, copying, or pasting. If `false`, trim rec does not remain after cutting, copying, or pasting.	`true` or `false`	`false`
trimRectSizing-Handles	Specifies whether the trim rec is sizeable (`true`).	`true` or `false`	`true`
uIClassID	Specifies the name of the L&F class that renders this component.		`PanelUI`
unicodeData-StreamEnabled	Specifies whether the session can receive Unicode data fields sent by a host. If `true`, the session can receive Unicode data field sent by a host. If `false`, the session cannot receive Unicode data field sent by a host	`true` or `false`	`false`
userID	Specifies the user ID that is used in the SSH authentication is processed either with the public-key or password.	Valid user ID	
userPassword	Specifies the user password that is used in the SSH authentication process.	Valid user password	
useSSHPublicK-eyAuthentication	Specifies whether the SSH public key authentication is enabled (`true`).	`true` or `false`	`false`
valid	Specifies whether the component is valid. A component is valid when it is correctly sized and positioned within its parent container and all its children are also valid.	`true` or `false`	`false`
validateRoot	Specifies that the entire tree beginning with the root is validated.	`true` or `false`	`false`

Table 17. Display property verification point properties (continued)

Property	Description	Valid values	Default value
verifyInputWhen-FocusTarget	Specifies whether the input verifier for the current focus owner is called before the component requests focus.	`true` or `false`	`true`
version	Specifies the version.		
visible	Specifies that the component is visible when set to `true`.	`true` or `false`	`true`
visibleRect	Specifies the `visible rectangle` of the component. The intersection of the component's visible rectangle and all of its ancestors' visible rectangles.		`java.awt-.Rectan-gle[x=0,y=0,width=740,height=`
width	Specifies the width of the component.		`740`
workstationID	Specifies the workstation ID that is used during enhanced negotiation for 5250. **Note:** For best results, call the isValidWorkstationID (String workstationID) function to check the validity of the string. All lowercase characters are converted to uppercase.		
workstationI-DReady	Specifies whether the workstation ID is ready. This method is only valid for 5250 session. If `true`, the workstation ID is ready. If `false`, the workstation ID is not ready.	`true` or `false`	`false`
x	Specifies the current x coordinate of the origin of the component.		`0`
y	Specifies the current y coordinate of the origin of the component.		`0`

Properties of operator information area (OIA) property verification points

You can create OIA property verification points to test the OIA in your application. When you record the verification point, a baseline of the OIA is created. Every time you play back the script, the OIA is compared to check if any changes have occurred. This helps in identifying any mismatch in OIA.

Table 1 shows the properties, descriptions, and default values of OIA property verification points.

Table 18. OIA property verification points properties

Property	Description	Default Value
alphanumeric	Specifies whether the input is limited to alphanumeric characters.	`true`
class	Specifies the Rational® Functional TesterHCL OneTest™ UI class name. For example, `HtmlTable` is the class name for a <Table> element.	`com.ibm.eNet-work.ECL.ECLOIA`

Table 18. OIA property verification points properties (continued)

Property	Description	Default Value
commCheckCode	Specifies the communication check code.	`0`
inputInhibited	Specifies whether the input is inhibited.	`0`
machineCheckCode	Specifies the machine check code.	`0`
numeric	Specifies whether the host field is limited to numeric input.	`false`
oIAEventDelay	Specifies the OIA event delay.	`0`
progCheckCode	Specifies the program check code.	`0`

Properties of operator information area (OIA) data verification points

You can create OIA data verification points to test the OIA data in your application. When you record the verification point, a baseline of the OIA data is created. Every time you play back the script, the OIA data is compared to check if any changes have occurred. This helps in identifying any mismatch in OIA data.

Table 1 shows the properties, descriptions, and default values of OIA data verification points.

Property	Description	Default value
INHIBIT_NOTINHIBITED	Specifies whether the OIA is inhibited.	`INHIBIT_NOTIN-HIBITED`
INHIBIT_COMMCHECK	Specifies whether the COMMxxx check is inhibited.	
INHIBIT_SYSTEMWAIT	Specifies whether the SYSTEM check is inhibited.	
INHIBIT_MACHCHECK	Specifies whether the MACHxxx check is inhibited.	
INHIBIT_PROGCHECK	Specifies whether the PROGxxx check is inhibited.	
INHIBIT_OTHERINHIBIT	Specifies whether the OTHERINHIBIT check is inhibited.	
STATE_A_ONLINE	Specifies whether the session is online with a non-SNA connection.	`true`
STATE_COMM_CHECK	Specifies whether a COMM check is performed.	`false`
STATE_COMM_ERR_REM	Specifies whether to display a communications error reminder.	`false`
STATE_CONTROLLER_READY	Specifies whether the controller is in a ready state.	`true`
STATE_DO_NOT_ENTER	Specifies the state of the do not enter mask.	`false`
STATE_ELSEWHERE	Specifies whether the keystroke is in the wrong place on the screen and that the cursor must be moved.	`false`
STATE_ENCRYPT	Specifies whether the session is encrypted.	`false`
STATE_FN_MINUS	Specifies whether the function is currently available.	`false`
STATE_GR_CURSOR	Specifies the graphic cursor state.	`false`

Property	Description	Default value
STATE_INPUT_ERROR	Specifies whether there has been operator input error.	`false`
STATE_INSERT	Specifies the ECL insert state.	`false`
STATE_MORE_THAN	Specifies whether too many characters are typed into the field.	`false`
STATE_MSG_WAITING	Specifies the state of the message-waiting indicator.	`false`
STATE_MY_JOB	Specifies whether the session is connected to a host application.	`false`
STATE_OP_SYS	Specifies whether the session is connected to SSCP (SNA).	`false`
STATE_PROG_CHECK	Specifies whether a program check (error in the data stream) has occurred.	`false`
STATE_SYM_MINUS	Specifies whether the entered symbol is available.	`false`
STATE_SYS_LOCK	Specifies the state of the system lock after the AIM is key pressed.	`false`
STATE_TIME	Specifies whether the keyboard is inhibited.	`false`
STATE_UNOWNED	Specifies whether the session is connected.	`false`
STATE_WHAT_KEY	Specifies whether the keystroke is valid at this time.	`false`

Creating character verification points

You can create a character verification point to test the properties of a certain character in your application. The character is a part of a host text field (ECLField). The character can be in any host text field that is identified by the Rational® Functional TesterHCL OneTest™ UI. When you record the verification point, a baseline of the character is created. Every time you play back the script, the character is compared to check if any changes have occurred. This helps in identifying any mismatch in character.

1. Log in and interact with the host while creating a host connection script. For more information on creating a host connection script, see the related topics.
2. When you are in the window in which you want to perform a test on an individual character, click the **Verification Point and Action Wizard**  icon in the Recording monitor window.
3. On the Select an Object page of the wizard, click **Selection Method > Test Object Browser**.
   Result
   A hierarchical tree of testable objects in your application is displayed in the Test Object Browser.

   Note: : You cannot use the drag hand with the corresponding **Object Finder** button to set a verification point on a single character.

4. Navigate the tree to locate the object that corresponds to the character that you want to test. Select it to display its recognition properties. To navigate the tree:

a. From the Select Test Object pane, click **Frame > SplitPane > Terminal > Screen**.

b. Expand the field that contains the character that you want to test. Fields are named *Field_x_y*, where *x* is the row that contains the field and *y* is the column. On Virtual Terminal (UNIX) systems, there is only one row.

c. Select the character that you want to test. Characters are named *Char_x_y_z*, where *x* and *y* are the same as the corresponding field, and *z* is the character position within the field.
Result
The recognition properties of this character are shown in the **Object Recognition Properties** pane.

The value of the *char* property is the character as it is displayed:
- startRow: Starting row of the containing field
- startCol: Starting column of the containing field
- position: Position from beginning of the containing field
- class: Class name
- char: Character in the current position

Note: You can use this information to confirm that you have selected the correct character object. If no information is listed, the object cannot be tested or the environment might not be enabled for testing.

5. After you verify that you have selected the correct character, click **Next**.
Result
You can set a data verification point on the character value or set one or more of the following property verification points: background, blink, char, class, foreground, position, reverse, startCol, startRow, and underline.

Creating row verification points

You can create a row verification point to test the properties of a certain row in your application. The row is a part of a ECL screen. When you record the verification point, a baseline of the row is created. Every time you play back the script, the row is compared to check if any changes have occurred. This helps in identifying any mismatch in row.

Before you begin

To capture row verification points while recording the scripts for testing VT terminal-based applications, you must set the flag `rational.test.ft.fte.playback.vt_row_vp = true` in the ivory.properties file available at `<product installation directory>\FunctionalTester\bin` location. By default, this flag is set to "false".

1. Log in and interact with the host while creating a host connection script. For more information on creating a host connection script, see the related topics.
2. When you are in the window in which you want to perform a test on an individual row, click the **Verification Point and Action Wizard** icon in the Recording monitor window.

3. On the Select an Object page of the wizard, click **Selection Method > Drag Hand Selection**.
   Result
   Use the drag hand with the corresponding **Object Finder** button to set a verification point on a single row.

   Notes:
   - The **Drag Hand Selection** method is not available on Linux environments such as Ubuntu and Red Hat Enterprise Linux (RHEL). You must use the **Test Object Browser** method on Linux environments.
   - On Windows computer, you can also use the **Test Object Browser** method to capture verification points.
   - The row verification is valid only for virtual terminals and not for TN3270 and TN5250 terminals.

4. After you verify that you have selected the correct row, click **Next**.
   Result
   You can set a data verification point on the row value or set one or more of the following property verification points: char, class, length, rowEnd, rowStart, screenCols, screenRows, startRow, updatedLength.

Logging window content

You can log the contents of a window in a correctly formatted log file. The log contains the record of events that occur when recording the script.

1. Replace the default `New Script Helper Superclass` with the `TerminalScript`file.
   The replacement file logs contents of the window in a correctly formatted log file. The file is available in the examples folder in the Extension for Terminal-based Applications installation directory.
2. Add the following command to the script for each window that you want to add to the log file:
   `logInfo(formatScreenText(Screen(),true));`.
   For additional information see the following Web page: http://com.rational.test.ft.help/ChangingDefaultScriptHelperSuperclass.htm.

Extension for Terminal-based Applications states

Rational® Functional TesterHCL OneTest™ UI tracks the state of Extension for Terminal-based Applications sessions as you record a script and interact with the host session. To determine the state of the session, Rational® Functional TesterHCL OneTest™ UI checks whether the host session has focus. If the host session does not have focus, it waits for 0.5 second to see whether it gains focus. If the host session does not gain focus during that time, it returns the focus state to the frame. If the host session does have focus after the wait, Rational® Functional TesterHCL OneTest™ UI queries the synchronization code for the state of the terminal.

Extension for Terminal-based Applications session has one of the following three states:

Table 19. Session states

State	Description
UNINITIALIZED	The host window is not ready for interaction, because it is still updating, changing, or loading.
LOADED	The host window has finished updating, but the operator information area (OIA) is locked, usually because of invalid input.
READY	The host window is ready for interaction, has finished updating, and the OIA is unlocked.

The synchronization code determines the state based on synchronization algorithms. The algorithm depends upon the connection type.

Synchronization algorithms

Extension for Terminal-based Applications has three synchronization algorithms to determine the state of the terminal. The state of the terminal depends on the loading of presentation space.

The state of the terminal is recorded while you record a script. When the script is played back, Rational® Functional TesterHCL OneTest™ UI must wait for the state of the terminal to match the state during recording. The state must be the same in order to avoid sending commands to the host before the host is ready to receive input. The state of the terminal is determined by the operator information area (OIA) status (either locked or unlocked), which depends on the loading of presentation space. The loading of the presentation space is not an instantaneous process and different connection types load the presentation space differently.

The following algorithms gauge the state of the terminal:

- Default synchronization algorithm
- 3270 enhanced synchronization algorithm
- 5250 synchronization algorithm

The default wait period values that the synchronization algorithms use are as follows:

Algorithm	Synchronization settings
Default synchronization and 5250 synchronization	• Timeout (in milliseconds): 1200 • OIA State Timeout (in milliseconds): 300000 • Polling Interval (in milliseconds): 100
TN3270E Synchronization	Minimum time to wait (in milliseconds): 250

You can change any of these values using the Preferences window.

Default synchronization algorithm

The default synchronization algorithm starts when an attention identifier (AID) key is pressed. An attention identifier (AID) key is any key that triggers a presentation space update. Initially, the state of the terminal is `UNINITIALIZED`. The algorithm waits for a period of time for updates to the presentation space. You can change the wait time in the **Timeout** field in the Preferences window. The default wait time is 1200 milliseconds.

If Timeout is set to 1200 milliseconds, and an update occurs during the last 600 milliseconds, the algorithm waits for an additional 600 milliseconds for further updates. If, during this additional wait period, another update occurs during the last 300 milliseconds, the algorithm waits again for another 600 milliseconds for further updates. This continues until no updates are received during the last half of the last additional time period.

At this point, the state of the terminal is either `LOADED` (keyboard locked) or `READY` (keyboard unlocked), depending upon the OIA status.

3270 enhanced synchronization algorithm

Initially, the state of the terminal is `UNINITIALIZED`. The terminal state is not initialized for a minimum wait time. You can change the wait time in the **Minimum Wait Time** field in the Preferences window. The default value is 250 milliseconds.

The server notifies the algorithm that the presentation space updates are sent. The synchronization algorithm waits for the period specified in the **Minimum time to wait** field in the Preferences window, and then queries the state of OIA to determine whether or not to report `READY`. If the synchronization algorithm has waited for the period in the **Timeout** field in the Preferences window without the OIA state becoming ready, the algorithm reports a state of `LOADED`. At all other times, it reports a state of `UNINITIALIZED`. This algorithm requires that the correct service level for the TN3270 server is installed on the host.



Note: Communications Server for z/OS® 1.2 and later, introduced a new function called "contention resolution". If you do not have the latest maintenance levels for Communications Server for z/OS, you might experience COMM655 errors or endless loop conditions when trying to connect to a z/OS host. In such cases, configure your connection configurations so that they do not use contention resolution.

To change the contention resolution setting: In the Extension for Terminal-based Applications window, open the Advanced Settings window by clicking **Advanced**. Scroll to `negotiateCResolution` property in the **Configure optional advanced settings** list. Change the setting for this property from `true` to `false`.

5250 synchronization algorithm

Presentation space update events occur only once for 5250 sessions and not in groups as in 3270. The 5250 algorithm operates like the Default synchronization algorithm with one exception: When an update occurs, the state changes immediately to `LOADED` or `READY`, depending on the OIA state.

When you play back the script, Rational® Functional TesterHCL OneTest™ UI waits for the host terminal to show an appropriate state before it continues to run commands from the record script.

You can change the time setting in the Preferences window.

Note: Synchronization algorithms might not work all of the time with the specified time values, especially in dealing with hosts that have long network delays. If synchronization does not work with a certain part of your application you can insert manual sleep timers into the script to adjust the timing aspect or use manual synchronization. For more information, see the related topics.

Playing back host connection script

You can play back all your recorded actions, such as starting an application, the actions you perform, and stopping the application. There are several prerequisites to meet before you can reliably play back a script

Before you begin

To ensure that the script runs smoothly, perform the following prerequisites before playing back your script:

- Add sleep timers to the script if your host application has windows that take an abnormally long time, or your network connection is slow. Sleep timers cause the play engine to pause before sending input to the host or before trying to perform a test on the slow host window.
- Your actions, such as typing, clicking the mouse, and pressing buttons, are recorded into the script. If you have interacted with the host application prior to the host application being initialized, see Interacting with the host using the keyboard on page 1004.
- If your host application uses keys other than the function keys to cause the host window to change, Rational® Functional TesterHCL OneTest™ UI might not recognize the keys as host aid keys. For more information, see Using host key aids on page 1004.

1. Save changes in your script.
2. Click the **Run Functional Test Script** icon on the toolbar.

Adding manual sleep timers

When the host application moves from window to window, the transition is not instantaneous. Therefore, any input that needs to be sent to the host, such as typing text, pressing host aid keys, or testing verification points, needs to wait for the host window to become ready to receive input. Extension for Terminal-based Applications can be used to figure out the readiness of host screen for input.

About this task

Manually add sleep timers in the script where the host is making the transition from one window to the next before you play back a script. Insert the following command into the script:

```
        // add sleep timers during slow screen transitions
        sleep(5);
```

This command makes the playback pause for 5 seconds before moving to the next line in the script.

Add a sleep statement in the script to avoid the timeout of a synchronization algorithm before the page is actually loaded. Insert the following into the script:

```
sleep(10);
TFrame().inputKeys("logoff{ENTER}");
```

This prevents Rational® Functional TesterHCL OneTest™ UI from sending keystrokes to the application before the application is ready to receive them.

Important: Choose a sleep time that is appropriate for your connection. A sleep time that is set too short might cause problems when playing back scripts, because Extension for Terminal-based Applications might try to send commands to the host before it is ready to receive them or might try to check a verification point before the window has finished its transition. A sleep time that is set too long can affect performance.

Correcting object states

It is not always required to use the object state information that is captured when an object is manipulated in a script. If the state of the object does not match the state information in the script, the code might not work properly. If the exact state of the object is not necessary for the test that is performed, it might be beneficial to remove this state information.

The script for typing keystrokes, such as typing `logoff`, and then pressing Enter, when creating a host connection script is as follows:

```
 TFrame().inputKeys(logoff{ENTER});
```

Result

At playback time, Extension for Terminal-based Applications waits for the TFrame to be in the ready state before it sends the keystrokes.

The script for typing keystrokes before the synchronization algorithm for the terminal determines whether the presentation space is loaded when recording the script is as follows:

```
TFrame(ANY, UNINITIALIZED).inputKeys(logoff{ENTER});
```

This shows that the TFrame object was not initialized and not in the ready state when you started typing the command. At playback time, those keystrokes are sent as soon as the script reaches that line, regardless of the state of the terminal.

To correct the state of the terminal, remove the state information from the object in the script:

```
        //Remove "(ANY,UNINITIALIZED)" is any
        //TFrame(ANY,UNINITIALIZED).inputKeys(logoff{ENTER});
        TFrame().inputKeys(logoff{ENTER});
```

Using host aid keys

The most common host aid keys are Enter, and all the function keys. If the key is captured within curly braces {} in a script, the key acts as a host aid key.

About this task

Ensure that Rational® Functional TesterHCL OneTest™ UI pauses after sending the keystrokes whenever you press keys that cause the host window to change. Most function keys cause the host window to change. Rational® Functional TesterHCL OneTest™ UI records these keystrokes as separate commands to ensure that it pauses after sending a function key to the host.

```
TFrame().inputKeys("{F12}");
        TFrame().inputKeys("{F3}");
        TFrame().inputKeys("logoff{ENTER}");
```

If your host application uses other keys to cause the host window to change, manually separate pressing those keys into separate commands before playing back your scripts. Rational® Functional TesterHCL OneTest™ UI does not recognize these keys as host aid keys.

Exemple

For example, If `Tab` causes the host window to change, the following command will cause problems when the script is played back:

```
TFrame().inputKeys("{TAB}SomeHostCommand{ENTER}");
```

Separate the command after the first host aid key:

```
TFrame().inputKeys("{TAB}");
TFrame().inputKeys("SomeHostCommand{ENTER}");
```

Using manual synchronization

Use manual synchronization when performance is considered as higher priority than running the script automatically. The synchronization algorithms are reliable, but take more time to run than is necessary.

1. On the Recording Monitor toolbar, before you enter text, click the **Verification Point and Action Wizard**  icon.
2. Drag the **Object Finder**  over the host terminal to select the window object. A thick red line outlines the terminal when selected.
3. Click **Wait until**, and type the text input. The entry in the script is as follows:

   Result

   ```
   Screen().waitForExistence();
   TFrame().inputKeys("logoff{ENTER}");
   ```

   In this case, the correct window is loaded because of the wait time. You do not need to rely on the synchronization algorithm.
4. Change the second line of the previous script:

   ```
   TFrame(ANY, UNINITIALIZED).inputKeys("logoff{ENTER}");
   ```

   You can send the keystrokes regardless of the state of the terminal determined by the synchronization code.

LOADED, not READY

There are special cases during application testing when the window has finished loading, but the application inhibits keyboard input. For example, sometimes when you type invalid input, the host application locks the keyboard. You cannot continue until you reset the window.

In this case, the synchronization algorithm specifies a state of `LOADED` rather than a state of `READY`. If you need to interact with the terminal when it is in the `LOADED` state, type the following in the script:

```
TFrame(ANY, LOADED).inputKeys("logoff{ENTER}");
```

Printing a host session window

You can print your host session window.

About this task

Click **Print Screen** to print your host session screen.

Programmatic screen scraping for Terminal-based applications

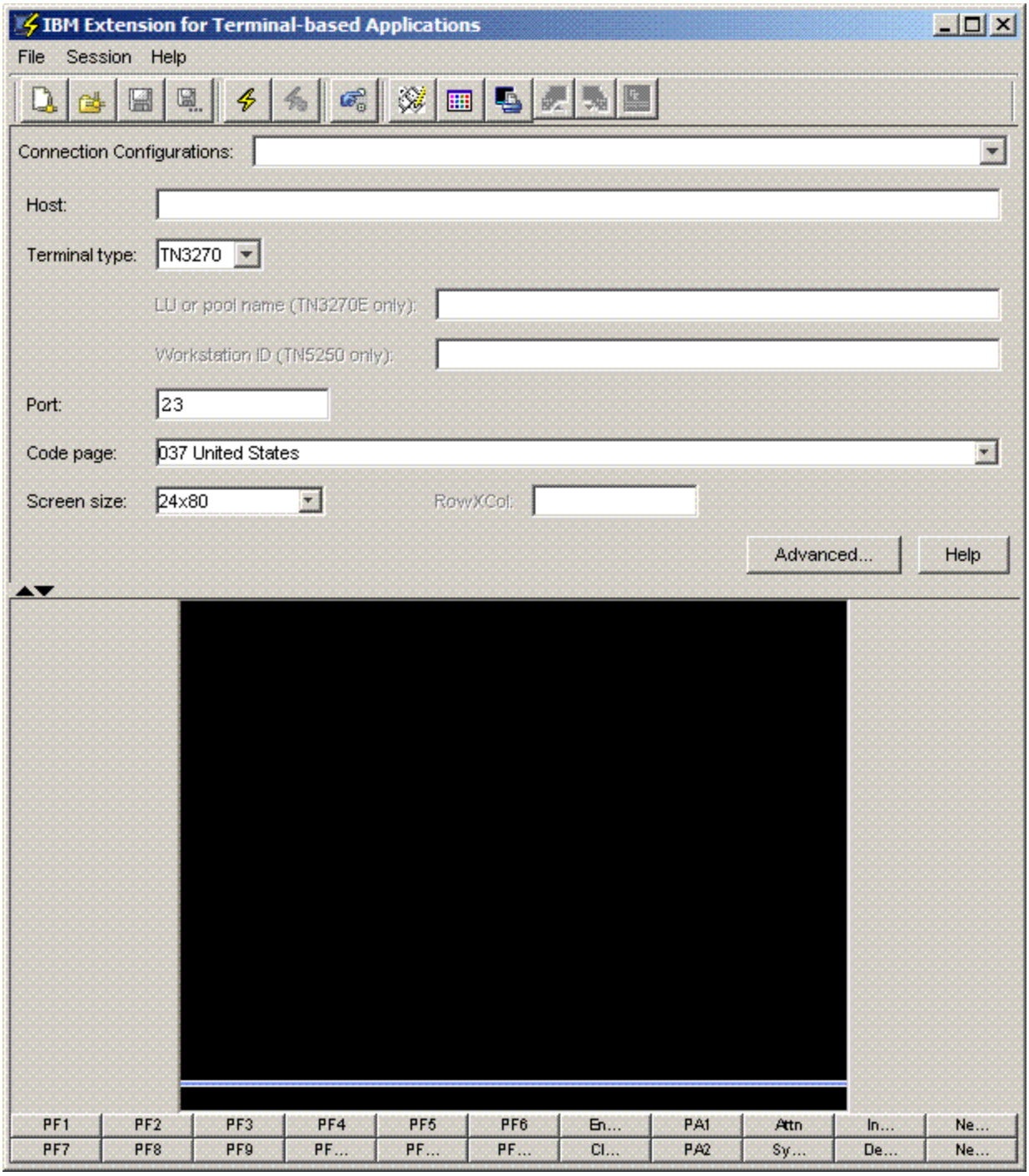
The Rational® Functional TesterHCL OneTest™ UI Extension for Terminal based applications supports programmatic screen scraping. In earlier versions of the Extensions, data verification on a terminal screen was performed only on fields, characters, or rows (virtual terminal). With programmatic screen scraping, data verification is performed on arbitrary locations. Programmatic screen scraping is used to scrape a portion of the terminal screen to determine whether the text is displayed at the specified position of the screen. Also, programmatic scraping makes the playback process wait to allow the text to appear on the screen. APIs are developed to support this feature.

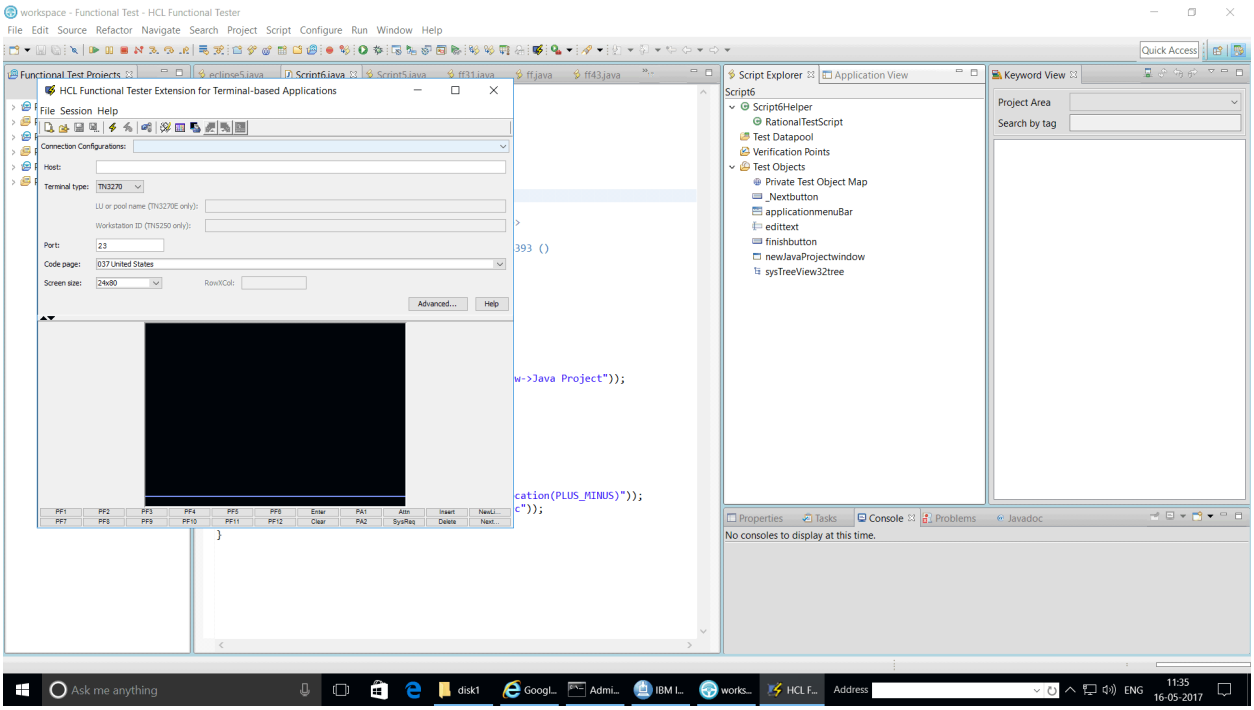
These methods are used to extract information from a portion of the screen:

Methods	Description
`public void startEventMonitor()`	Collects the screen changes and stores the changes in a buffer
`public boolean checkForChange(int startRow, int startCol, int endRow, int endCol, String text)`	Checks for the text at a particular position
`public boolean checkForChange(int startRow, int startCol, int endRow, int endCol, String text, boolean clearBuffer)`	Checks for the text at a particular position, and then clears the buffer
`public boolean waitForChange(int startRow, int startCol, int endRow, int endCol, String text)`	Waits for the text to be displayed in the specified position
`public boolean waitForChange(int startRow, int startCol, int endRow, int endCol, String text, long timeout)`	Waits for the text to be displayed in the specified position at the specified time
`clearHistory()`	Clears the buffer
`public void stopEventMonitor()`	Stops collecting input from the screen

Extension for Terminal-based Applications window

Use the Extension for Terminal-based Applications window to connect to the host and record or play back host interactions in the scripts.





The top pane of the window shows the connection configuration information. The bottom pane is the actual terminal view. You can move the divider up or down to increase the space of either portion of the window. The terminal session function keys that correspond to the function keys on the keyboard are located at the bottom of the window.

Extension for Terminal-based Applications toolbar uses the icons shown in the Table 1:

Table 20. Extension for Terminal Based Applications toolbar icons

Icon	Icon name	Description
	New Connection	Creates a new connection file. Clears the fields in the top pane of the window and resets them to default values.
	Open	Opens an existing connection file. Displays the **Open file** dialog box to reload the saved configuration file.
	Save	Saves current connection settings file so that you can reload the settings later.
	Save As	Saves as connection file. Provide a file name.
	Connect using the Current Connection Properties	Connects to a host using the specified configuration information.

Table 20. Extension for Terminal Based Applications toolbar icons (continued)

Icon	Icon name	Description
	**Extension for Terminal-based Applications**	Sets synchronization settings.

Configuring basic connection properties

You can configure the basic connection properties such as host, terminal type, port, code page, and screen size.

1. From the toolbar, click  icon to open a connection properties file, and select a connection configuration (.comm) file to connect to the host.
2. Type the name or IP address of the host computer.
3. Configure the terminal type:

    a. Select the type of terminal emulation connection to use: TN3270, TN3270E, TN5250, VT default, VT100, VT420-7, VT420-8 or VT UTF8.
    TN3270 terminal emulation provides connections to an IBM zSeries® (System 390) host. TN5250 terminal emulation typically provides connections to an iSeries™ (AS/400®) host. VT terminal emulation provides connections to a UNIX-like system. The default is TN3270.

    Note: Communications Server for z/OS® 1.2 and later introduced a new function called contention resolution. This new function on the client side was introduced in Host On-Demand 7.0.2. If you do not have the latest maintenance levels for Communications Server for z/OS, you might experience COMM655 errors or other endless loop conditions when you try to connect to a z/OS host. The default setting for the HOD layer is to request RFC 2355 Contention Resolution. If you experience such problems, configure the HOD layer so that it does not request RFC 2355 Contention Resolution.

    b. **Optional:** Click **Advanced** to change the contention resolution setting. Scroll down the properties in the **Configure optional advanced settings** list. Change the setting of `negotiateCResolution` to **false**. This applies to the TN3270E terminal only.

    c. Type the name of the specific Workstation ID on the host. This field is available only if you are using TN5250 terminal emulation.
    Leave this field blank if you are not sure about the Workstation ID to connect to.

    d. Type the name of the specific logical unit (LU) or LU pool on the host. This field is available only if you are using TN3270 Enhanced (TN3270E) terminal emulation.
    Leave this field blank if you are not sure about the LU to connect.

4. Type the port number on which the host is configured.

5. Select the code page that corresponds to the language that you are using. The default is **037 - United States**.
6. Select the number of horizontal and vertical characters that can be displayed in the screen size. The default is **24x80**.

Screen size options

You can change the size of the screen to have different numbers of lines and columns, depending on the host system. More lines and columns provides more data on each screen, while fewer lines and columns can be easy to read.

Table 1 shows only the available built-in screen sizes. You can customize the screen size. For information on customizing screen size, see related topics.

Table 21. Screen size options

Value	5250	3270	VT
24x80 (Default)	X	X	X
32x80		X	
36x80			X
43x80		X	
48x80			X
72x80			X
24x132			X
27x132	X	X	
36x132			X
48x132			X
72x132			X

Related information

Customizing screen size when connecting to a TN3270 or TN3270E host on page 962

Code page options

A code page is a particular assignment of code points to graphic characters. Make sure the code page that is used to connect to the host is supported by the host system, and is the one you want to use. The table contains code page options by language and by country/region.

3270 and 5250 code page options

Table 22. Code page options for 3270 and 5250

Value	Country or region
037	United States (**Default**)
037	Belgium
037	Brazil
037	Canada
037	Netherlands
037	Portugal
273	Germany
273	Austria
274	Belgium (Old)
275	Brazil (Old)
277	Denmark
277	Norway
278	Finland
278	Sweden
280	Italy
284	Spain
284	Latin-America (Spanish)
285	United Kingdom
297	France
420	Arabic
424	Hebrew (New)
500	Multilingual
803	Hebrew (Old)
838	Thai
870	Romania
870	Bosnia/Herzegovina
870	Croatia

Table 22. Code page options for 3270 and 5250

(continued)

870	Czech
870	Hungary
870	Poland
870	Slovakia
870	Slovenia
871	Iceland
875	Greece
930	Japanese (Katakana Extended)
930	Japanese (Katakana)
933	Korea (Extended)
937	Taiwan (Traditional Chinese Extended)
939	Japan (Latin Extended)
1025	Russia
1025	Belarus
1025	Bulgaria
1025	FYR Macedonia
1025	Serbia/Montenegro (Cyrillic)
1026	Turkey
1047	Open Edition
1112	Latvia
1112	Lithuania
1122	Estonia
1123	Ukraine
1137	Hindi
1140	United States Euro
1140	Belgium Euro
1140	Brazil Euro
1140	Canada Euro
1140	Netherlands Euro

Table 22. Code page options for 3270 and 5250
(continued)

1140	Portugal Euro
1141	Germany Euro
1141	Austria Euro
1142	Denmark Euro
1142	Norway Euro
1143	Finland Euro
1143	Sweden Euro
1144	Italy Euro
1145	Spain Euro
1145	Latin America Euro
1146	United Kingdom Euro
1147	France Euro
1148	Multilingual Euro
1149	Iceland Euro
1153	Romania Euro
1153	Bosnia/Herzegovina Euro
1153	Croatia Euro
1153	Czech Republic Euro
1153	Hungary Euro
1153	Poland Euro
1153	Slovakia Euro
1153	Slovenia Euro
1154	Russia Euro
1154	Belarus Euro
1154	Bulgaria Euro
1154	FYR Macedonia Euro
1154	Serbia/Montenegro (Cyrillic) Euro
1155	Turkey Euro
1156	Latvia Euro

Table 22. Code page options for 3270 and 5250 (continued)

1156	Lithuania Euro
1157	Estonia Euro
1158	Ukraine Euro
1160	Thai Euro
1364	Korea Euro
1371	Taiwan (Traditional Chinese) Euro
1388	PRC (Simplified Chinese; GBK)
1390	Japanese (Katakana Unicode Extended) Euro
1399	Japanese (Latin Unicode Extended) Euro

Virtual Terminal (VT) sessions code page options

Table 23. Code page options for VT sessions

Value	Language
874	Thai
935	Simplified Chinese
937	Traditional Chinese
1011	German
1012	Italian
1020	French Canadian
1021	Swiss
1023	Spanish
1100	United States (English)
1101	British (English)
1102	Dutch
1103	Finnish
1104	French
1105	Norwegian/Danish
1106	Swedish
1208	UTF-8

Remapping keyboard

You can remap a key or a combination of keys to a character or host function using the Keyboard Remap window. You can save your keyboard remapping to a file and load it to use later. You can also turn off key repetition when a key is held down.

Remapping key to a character

You can remap a key to a character that is not available by default from the keyboard.

1. In the Extension for Terminal based Applications window, click **Keyboard Remap** .
2. To locate and load the remapping file if you have saved the remapped keys to a file, click **Load**.
3. Click **Key Assignment** tab.
4. Click **Characters** from the **Category** list.
5. Select a character to which you want to remap a given key or combination of keys from the list of characters.
6. Click **Assign a Key**.
7. Press the key or key combination that you want to remap to the character on your keyboard. For example, to use Alt+4, press and hold the keys down simultaneously for key combinations.
8. Save keyboard remapping.
    a. In the Keyboard Remap dialog box, click **Save**.
    b. Type a file name or select the name of an existing file in the **File name** field.
    c. Click **Save**.
9. Click **OK**.

Remapping key to host function

You can remap a key to a host function that is not available by default from the keyboard or remap a combination of keys to a host function, including the Shift, Alt, and Ctrl keys. You can remap a key to a host function because the default key values are not always ideal for all host applications.

About this task

For example, in some panel-driven z/OS® application programs, it is convenient to have a function key to erase the contents of a field, such as `Erase EOF`. This function is not provided by default, but it can be enabled by remapping a function to a key or combination of keys of your choice.

Note: Remapped keys retain their custom values during the host session for which the keys were remapped only. The remapping does not persist across subsequent host sessions that are started from other host connection windows.

1. In the Extension for Terminal-based Applications window, click **Keyboard Remap**.
2. **Optional:** To locate and load the remapping file if you have saved the remapped keys to a file, click **Load**.
3. Click **Key Assignment** tab.
4. Click **Host Functions** from the **Category** list.

5. Select the function to which you want to remap a given key or combination of keys from the list of host functions.
6. Click **Assign a Key**.
7. Press the key or key combination that you want to remap to the host function. For example, to use Alt+4, press and hold the keys simultaneously for key combinations.
8. **Optional:** Save keyboard remapping.
9. Click **OK**.

Loading keyboard remap from file

You can load keyboard remapping that you have created and saved in a file.

1. Click **Keyboard Remap**.
2. Click **Load**.
3. Locate and select the file name you want to load.
4. Click **Load**.
5. Click **OK**.

Results

Your keyboard is remapped according to the remapping saved in the loaded file.

Turning off key repetition

You can choose whether a key that you hold down produces a single keystroke or multiple keystrokes by using the key repetition function.

1. Click **Keyboard Remap**.
2. **Optional:** To locate and load a file that you have saved to a file, click **Load**.
3. Click the **Key Repetition** tab in **Keyboard Remap** dialog box.
4. Click **Add Key**.
5. Press the key that you want to be a non-repeating key.
6. **Optional:** Click the **Save** button to save turning off key repetition.
7. Click **OK**.

Remapping session screen colors

You can remap the colors of your host session. Color remapping does not persist for subsequent host sessions.

1. Make sure the host terminal is running and connected to the host.
2. Click **Color Remap**.
3. From the left pane of the Color Remap window, select the field for the color change.
4. Select the color that you want from the left pane.
5. Close the Color Remap window for the changes to take effect.

Sending files from workstation to a host system (3270 host sessions only)

You can send files from a workstation to a host. File transfer is available only for 3270 and 3270E sessions.

Before you begin

You must be connected to a host for the file transfer functions to work.

1. Log in to your host system and open the command line interface.
2. Before you transfer a file, check the default file transfer settings to make sure that they are correct. To set the default settings of your host system, click **Default settings for File Transfer**. Three types are supported in `Host Type`: MVS/TSO, VM/CMS and CICS®. Click **OK**.
3. Specify a PC file name, host file name, and transfer mode in the **Send Files to Host** window.
4. To add the specified files to the Transfer list pane, click **Add to List**.
5. **Optional:** To open a saved list, click **Open List**.
6. **Optional:** To save the files that are listed in the Transfer List window to a file, click **Save List**.
7. Select the files in the **Transfer List** window that you want to send to the host.
8. To send the selected files, click **Send**.

   Result

   The specified files are transferred.

Retrieving files from a host system to the workstation (3270 host sessions only)

You can retrieve files from a host system to the workstation. File transfer is available only for 3270 and 3270E sessions.

Before you begin

You must be connected to a host for the file transfer functions to work.

1. Log in to your host system and open the command line interface.
2. To set the default settings of your host system, click **Default settings for File Transfer**. Three types are supported in `Host Type`: MVS/TSO, VM/CMS and CICS®.
3. Click **OK**.
4. Specify a PC file name, host file name, and transfer mode from the **Receive Files from Host** window.
5. To add the specified files to the Transfer list pane, click **Add to List**.
6. **Optional:** To open a saved list, click **Open List**.
7. **Optional:** To save the files listed in the Transfer List window to a file, click **Save List**.
8. In the Transfer List window, select the files that you want to receive from the host.
9. To retrieve the selected files, click **Receive**.

   Result

   The specified files are transferred.

Connecting to a virtual terminal (UNIX) session

The host terminal panel shows the default values for a TN3270 terminal connection when you launch Extension for Terminal-based Applications. You can also connect to a VT (UNIX) session.

Before you begin

The host terminal must be running and have access to a VT session.

1. Connect to the host terminal.
2. Select one of the VT terminal types from **Terminal type** list.
   If you select the VT UTF8 terminal type, make sure that you also select a UTF8 code page. By default, the 1100 DEC Multinational Replacement Character Set code page and the 24 x 80 screen size are displayed in **Code page** and **Screen size** lists.
3. Specify the code page from the **Code page** list.
   If you are connecting to a VT UTF8 session, make sure you select a UTF8 code page.
4. Specify the screen dimensions from the **Screen size** list.
5. Click **Advanced** and verify that the entries in the **Advanced Setting** table have been updated with specific VT session properties and their default values.
6. Click **OK** to close the Advanced Settings dialog box.
7. Click **Connect** to connect to the host system.

Extension for Terminal-based Applications preferences

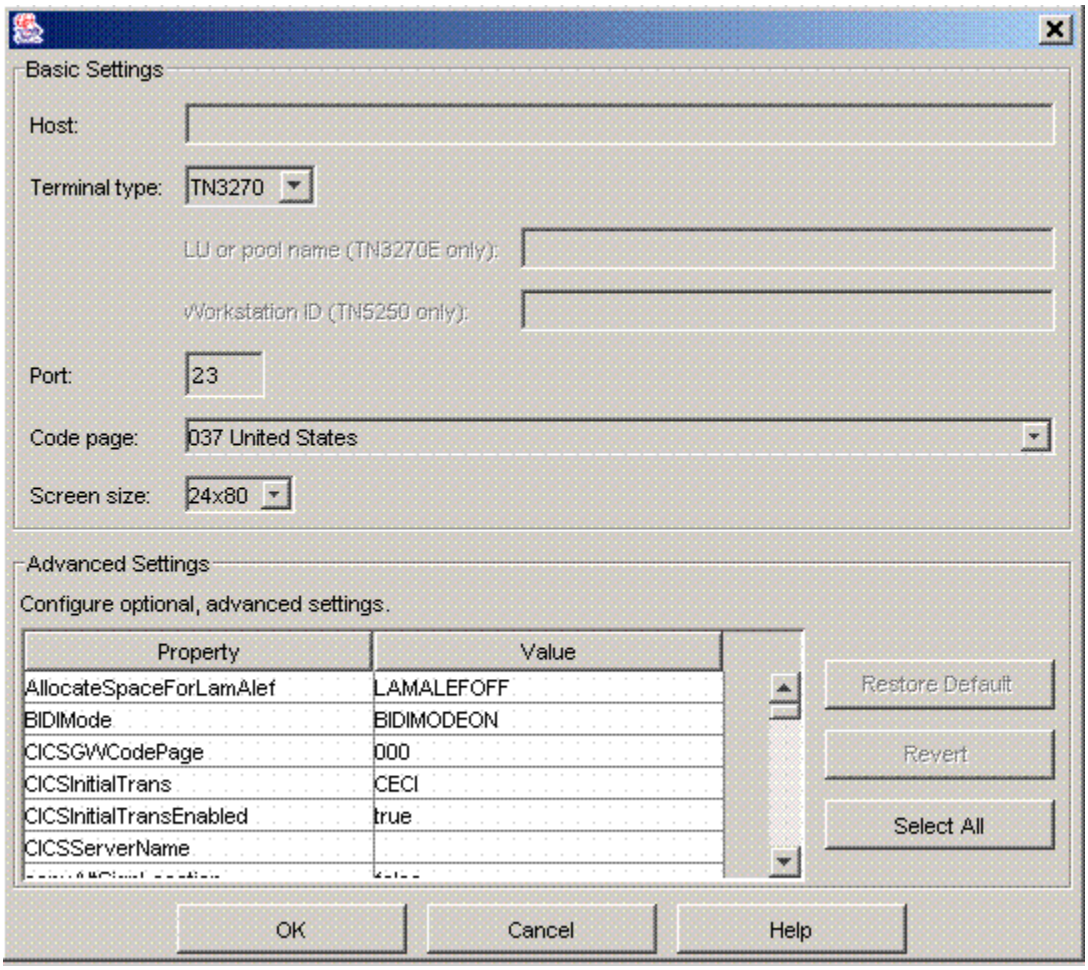
You can set the synchronization settings using Extension for Terminal-based Applications Preferences window.

Synchronization settings have the following fields for fine tuning synchronization algorithms:

Field	Description
Timeout	The wait time for the session to stabilize.
OIA State Timeout	The wait time for the operator information area to reach a stable state. The states are: UNINITIALIZED, READY and LOADED.
Polling Interval	The wait time interval between polls to determine the state of the session.
Minimum time to wait	The minimum wait time before starting to poll the session.
Default Folder for Connection Configuration Files	The path of the folder to store connection configuration files.

Advanced Connection Settings window

You can configure the basic settings and advanced settings using Advanced Connection Settings window.



The value of the property in optional and advanced settings can be changed by selecting the property and typing a new value.

The Advanced Connection Settings window has the following buttons:

Button	Description
Restore Default	Resets the selected advanced connection properties to default values.
Revert	Resets the selected properties values to the values that were in effect when you opened the Advanced Connection Properties window.
Select All	Selects all properties in the advanced connection properties list.
OK	Completes the connection configuration. Returns to Extension for Terminal-based Applications window and connects to host.

Advanced connection properties

You can configure the advanced connection properties.

Table 1 shows the properties, descriptions, valid values and default values of advanced connection properties.

Table 24. Advanced connection properties

Property	Description	Valid Values	Default Value
AllocateSpaceForLamAlef	Specifies whether LamAlef is expanded or compressed. This property applies to Arabic sessions only.		`LAMALEFOFF`
BIDIMode	Specifies whether to enable or disable bidirectional (BIDI) functions, like character shaping. This property applies to Arabic virtual terminal (VT) sessions only.	`BIDIMODEON` and `BIDIMODEOFF`	`BIDIMODEON`
CICSGWCodePage	Specifies the CICS® gateway.		`000`
CICSInitialTrans	Specifies the names of the initial transaction identifiers are strings between 1 and 128 characters. The string identifies the initial transaction and any parameters to be specified upon connection to the server. The transaction is the first four characters, or the characters up to the first blank in the string. The remaining data is passed to the transaction on its invocation.		`CECI`
CICSInitialTransEnabled	Specifies whether an initial transaction is started when a CICS gateway session is established.	`true` or `false`	`true`
CICSServerName	Specifies the CICS server name.		
connectionTimeout	Specifies the connection timeout value.		`0`
copyAltSignLocation	Specifies the mode of cut or copy for the sign of a number. `true` specifies a sign character is placed in front of the number. `false` specifies a sign character is placed in the same location relative to the number as it appears on the screen.	`true` or `false`	`false`
copyOnlyIfTrimmed	Specifies whether to set the copy error when there is no trim. `true` sets the copy error when no trim. `false` copies the entire screen when no trim.	`true` or `false`	`false`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
cursorDirection	Specifies whether cursor direction is left to right or right to left. This property applies to BIDI Visual VT sessions only.	`CURSOR_LEFTTORIGHT` or `CURSOR_-RIGHTTOLEFT`	`CURSOR_LEFT-TORIGHT`
CursorMovemen-tState	Specifies whether cursor movement within the presentation space by a mouse click is allowed. This property currently applies to VT sessions only.	`true` or `false`	`true`
ENPTUI	Indicates whether to enable the Enhanced Non-Programmable Terminal User Interface (ENPTUI) functionality. This property can be enabled for 5250 sessions only.	`true` or `false`	`false`
EntryAssist_bell	Enables or disables an audible signal when the cursor enters the column set for the End of Line Signal Column. `true` turns on bell. `false` turns off bell.	`true` or `false`	`false`
EntryAssist_bell-Col	Controls the column number at which you want the audible signal for the End of Line to be sounded. The audible signal will only sound if the EntryAssist_bell property is set to `true`.	Valid column numbers	`75`
EntryAssist_DOC-mode	Enables or disables the Entry Assist features. The Entry Assist (DOC mode) features make it easier to edit text documents in a 3270 display session. `true` turns DOC mode on. `false` turns DOC mode off.	`true` or `false`	`false`
EntryAssist_DOC-wordWrap	Enables or disables word wrap. When word wrap is enabled a word that is typed at the right margin is moved in its entirety to the first unprotected field in the next row, provided that the unprotected field has enough blank space to the left to contain the word. The area on the previous row vacated by the word is filled with spaces. If the unprotected field does not have enough blank space at the left to contain the word, then the word is not moved. The effect is the same as though word wrap were not	`true` or `false`	`true`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	enabled. When `true`, turns word wrap on and `false` turns word wrap off		
EntryAssist_endCol	Controls the right margin for DOC mode. When DOC mode is on, the rightmost cursor position in a row is the last unprotected character position to the left of the end column.	Valid column numbers	`80`
EntryAssist_startCol	Controls the left margin for DOC mode. When DOC mode is on, the leftmost cursor position in a row is the first unprotected character position to the right of the start column.	Valid column numbers	`1`
EntryAssist_tabstop	Controls how many spaces to skip when the Tab key is pressed.	Valid numbers of spaces	`8`
EntryAssist_tabstops	Controls the columns at which you want tab stops. When tab stops are set, pressing the tab key causes the cursor to skip to one of the following positions, whichever is first: • The next tab stop in the same unprotected field on the same row. (Tab stops cannot be defined outside the left or right margin.) • The first character position in the next unprotected field on the same row, if that character position is within the margins. • The first character position in the next unprotected field in a subsequent row, if that character position is within the margins. **Note:** Characters skipped as the result of a tab key are not set to blanks. When characters are in an unprotected field and the cursor skips over them because of pressing the tab key, they are not set to blanks. Only nulls that the	String representations of arrays of columns to use as tab stops. For example: `5`, `10`, `15`, `20`, `25`	

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	cursor skips as the result of a tab key are set to blanks.		
graphicsCellSize	Specifies the graphic cell size.		`0`
hostBackup1	Host name or IP address of the backup1 server. Displayed as the `Destination address` of backup1 on property panels. Applies to all session types.		
hostBackup2	Host name or IP address of the backup2 server. Displayed as as `Destination address` of backup2 on property panels. Applies to all session types.		
hostGraphics	Indicates whether to enable the host graphics function. This property can be enabled for 3270 sessions only.	`true` or `false`	
InsertOffOnAIDKEY	Sets the InsertOffOnAIDKEY property of Session. **Insert mode is set as follows** / **Any AID key performs as follows** `on` and InsertOffOnAIDKEY is `true` — Turns insert mode `off` `on` and InsertOffOnAIDKEY is `false` — Has no effect on the insert mode `off` — Does not turn insert mode `on` regardless of the state of InsertOffOnAIDKEY This property is valid for 3270 and CICS sessions only.	`true` or `false`	`false`
keyStoreFilePath	Specifies the path and name of the keystore file on the client workstation that contains the client public and private keys.	Valid path and file name of the keystore file	

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
keyStorePassword	The password that is required to open the keystore file on the client workstation.	Correct password to open the keystore file.	no password
lastHostWithoutTimeout			`true`
LUMLicensing	Specifies the license method.	`LUM` or `HOD`	`HOD`
LUMPort	Specifies the LUM port.	Valid port numbers	`80`
LUMServer	Specifies the LUM server name.	Valid LUM server names	
LUNameBackup1	The name of the LU or LU Pool, defined at the backup1 server, to which you want the session to connect. Displayed as `LU` or `Pool Name` of backup1 on property panels. Applies to 3270 Display and 3270 Printer session types.	Valid LU or LU pool names	
LUNameBackup2	The name of the LU or LU Pool, defined at the backup2 server, to which you want this session to connect. Displayed as `LU` or `Pool Name` of backup2 on property panels. Applies to 3270 Display and 3270 Printer session types.	Valid LU or LU pool names	
negotiateCResolution	Specifies whether to enable Negotiate Contention Resolution.	`true` or `false`	`true`
netName	The name of the terminal resource to be installed or reserved.		
numeralShape	Specifies the numeral shape as nominal, national or contextual for strings that are sent to the presentation space. This applies to Arabic hosts only.	`NOMINAL`, `NATIONAL`, or `CONTEXTUAL`	`NOMINAL`
numeralShapeDisp	Specifies how numerals are shaped. This property applies to Arabic VT sessions only.	`NOMINAL_DISP`, `NATIONAL_DISP`, or `CONTEXTUAL_DISP`	`CONTEXTUAL_DISP`
numericFieldLock	Specifies whether to limit the field characters of a session to numeric values. When true, only characters 0 through 9, -, +, period (.), and comma (,) are valid in fields that are defined by a host application as numeric. This property is valid for 3270 and CICS sessions only.	`true` or `false`	`false`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
numericSwapEnabled	Enables Numeric swapping. This property applies to Arabic 3270 sessions only.	`true` or `false`	`true`
panelOnlyTCPIPInactivityTimeout			`0`
pasteFieldWrap	Enables wrap on field. This property does not apply to VT sessions. `true` sets wrap on field. `false` sets normal wrap.	`true` or `false`	`false`
pasteLineWrap	Enables line wrap on field. `true` sets line wrap on field. `false` sets normal wrap.	`true` or `false`	`false`
pasteStopAtProtectedLine	Specifies whether to enable paste in a protected area. This property does not apply to VT sessions. `true`disables paste on a protected line. `false`enables normal paste.	`true` or `false`	`false`
pasteTabColumns	Specifies the pasteTabColumns to set the number of columns that are represented by a tab. If this option is active the input skips to the column that is a multiple of this setting.	Size of the tab in columns	`4`
pasteTabOptions	Specifies the pasteTabOptions.		`2`
pasteTabSpaces	Sets the pasteTabSpaces to the number of spaces that are represented by a tab. If this option is active, the input skips the number of spaces that is specified in this setting.	Number of spaces to advance for a tab	`1`
pasteToTrimmedArea	Specifies whether pasting is enabled in trimmed areas. This property does not apply to VT sessions. `true`sets paste to paste into trimmed area if defined. `false` sets paste to normal paste.	`true` or `false`	`false`
pasteWordBreak	Specifies whether paste splits words. This property does not apply to VT sessions. `true` sets paste to not split words. `false` sets paste to normal paste.	`true` or `false`	`true`
PDTFile	Specifies the name of a printer definition table (PDT) file. The PDT that you specify must be suitable for the printer and for the printer-em-		

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	ulation mode that the printer will use (such as PCL, PPDS. PostScript is not supported).		
portBackup1	The port number on which the backup1 server is configured. Displayed as `Destination port` of backup1 on property panels. Applies to all session types.		`23`
portBackup2	The port number on which the backup2 server is configured. Displayed as `Destination port` of backup2 on property panels. Applies to all session types.		`23`
printDestination	Specifies whether the output goes to a printer or to a file. `true` goes to printer. `false` goes to file.	`true` or `false`	`true`
printerName	Specifies the name of the destination printer device.	Valid print destination printers	`LPT1`
printFileName	Specifies the name to be assigned to the print file.	Valid print file names	
proxyAuthen-Method	Specifies the authentication method between the Host On-Demand session and proxy server. Select one of the following: • **Basic (HTTP only)**: The Host On-Demand session provides a user ID and password to the HTTP proxy server. • **Clear Text (SOCKS v5 only)**: The Host On-Demand session provides an unencrypted user ID and password to the socks proxy server. • **None**: The Host On-Demand session does not provide a user ID and password to the HTTP proxy or socks server. **Note:** If you select Basic or Clear Text as the proxy authentication method,		`SESSION_PROXY_AUTHEN_NONE`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	you must specify a User ID and Password.		
proxyServerName	Specifies the host name or IP address of the HTTP or socks proxy server.		
proxyServerPort	Specifies the TCP port number of the HTTP or socks proxy server.		`1080`
proxyType	Specifies the type of proxy server a host session uses. • Default browser Setting • HTTP Proxy • SOCKS v4 • SOCKS v5 • SOCKS v4, if v5 unavailable		`SESSION_PROXY_BROWSER_DEFAULT`
proxyUserID	Specifies the user ID that the Host On-Demand session provides to authenticate with the HTTP or socks proxy server.		
proxyUserPassword	Specifies the password that the Host On-Demand session provides to authenticate with the HTTP or socks proxy server.		
roundTrip	Specifies whether the roundTrip is in `on` or `off` mode. This method applies to bidirectional hosts only.	`ON` or `OFF`	`ON`
RTLUnicodeOverride	Enables or disables the RTL Unicode Override option. This applies to BIDI 5250 Hosts only.	`RTLUNICODEON` or `RTLUNICODEOFF`	`RTLUNICODEOFF`
SecurityProtocol	Specifies whether to use the TLS v1.0 protocol, the Secure Sockets Layer (SSL) protocol, or the Secure Shell (SSH) protocol for providing security. If set to TLS (default), and if the server is TLS-enabled, then a TLS v1.0 connection is provided. If the server is not TLS-enabled, then the server negotiates the connection down to SSL protocol.	`SESSION_PROTOCOL_TLS`, `SESSION_PROTOCOL_SSL`, `SESSION_PROTOCOL_SSH`	`SESSION_PROTOCOL_TLS`

Table 24. Advanced connection properties (continued)

Property	Description			Valid Values	Default Value
	Constant	Value	Description		
	SESSION_PROTOCOL_TLS	TLS	Use TLS v1.0 protocol (default)		
	SESSION_PROTOCOL_SSL	SSL only	Use SSL v3.0 protocol to provide security		
	SESSION_PROTOCOL_SSH	SSH	Use SSH protocol v2.0		
separateFiles	Specifies whether print files are stored in separate files. This property applies to Host On-Demand VT sessions only, 3270 printer, and 5250 printer sessions. When true, saves print jobs in separate files and false, saves print jobs in a single file.			true or false	false
serviceMgrHost	Specifies the name for the Host On-Demand server.				
SESSION_PROXY_AUTHEN_BASIC	Sets the authentication to basic when the connection goes through an HTTP proxy. This is not used when the proxyType is set to BROWSER_DEFAULT.				SESSION_PROXY_AUTHEN_BASIC
SESSION_PROXY_AUTHEN_CLEAR_TEXT	Sets the authentication to clear text when the connection goes through a SOCKS V5 proxy. This is not used when the proxyType is set to BROWSER_DEFAULT.				SESSION_PROXY_AUTHEN_CLEAR_TEXT
SESSION_PROXY_AUTHEN_NONE	Specifies that the session does not use a proxy server. This is not used when the proxyType is set to BROWSER_DEFAULT.				SESSION_PROXY_AUTHEN_NONE
SESSION_PROXY_AUTHEN_BROWSER_DEFAULT	Specifies that the session uses the proxy settings of the web browser. This is not used when the proxyType is set to SESSION_PROXY_BROWSER_DEFAULT.				SESSION_PROXY_BROWSER_DEFAULT
SESSION_PROXY_AUTHEN_HTTP	Specifies that the session connects only through an HTTP proxy server, overriding the proxy settings defined in the web browser.				SESSION_PROXY_HTTP

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	This is not used when the proxyType is set to `SESSION_PROXY_HTTP`.		
SESSION_PROXY_AUTHEN_SOCKS_V4	Specifies that the session connects through a SOCKS v4 proxy server only, overriding the proxy settings defined in the web browser. Socks version 4 proxy server connects to a host system on behalf of a Host On-Demand client and transmits data between the client and the host system. This is not used when the proxyType is set to `SESSION_PROXY_SOCKS_V4`.		`SESSION_PROXY_SOCKS_V4`
SESSION_PROXY_AUTHEN_SOCKS_V5	Specifies that the session connects through a SOCKS v5 proxy server only, overriding the proxy settings defined in the web browser. SOCKS v5 includes the complete functionality of SOCKS v 4 and in addition, it supports authentication to the proxy server, IP version 6 addressing, domain names, and other networking features. This is not used when the proxyType is set to `SESSION_PROXY_SOCKS_V5`.		`SESSION_PROXY_SOCKS_V5`
SESSION_PROXY_AUTHEN_SOCKS_V5_THEN_V4	Specifies that the session first attempts to connect using SOCKS v5. However, if the proxy server does not support SOCKS v5, the session connects using SOCKS v4. In either case, the session overrides the proxy settings defined in the web browser. Proxy Server Name and Proxy Server Port are unavailable if you select Use Default Browser Setting as the Proxy Type. This is not used when the proxyType is set to `SESSION_PROXY_SOCKS_V5_THEN_V4`.		`SESSION_PROXY_SOCKS_V5_THEN_V4`
sessionID	The short name that you want to assign to this session (displayed in the OIA). It must be unique to this configuration. Appears as "Session ID" on property panels. Applies to all session types. This is not used when the proxyType is set to `BROWSER_DEFAULT`.		
sessionName	Specifies the name of the session.		

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
showTextAttributesEnabled	Specifies the Show Text Attributes property. This property applies to logical BIDI VT sessions only.		`true`
SLPAS400Name	Connects a session to a specific iSeries™ server. Displayed as "AS/400® Name (SLP)" on property panels. Applies to 5250 Display and 5250 Printer session types. Use the fully-qualified SNA CP name (for example, USIBMNM.RAS400B).		
SLPEnabled	Specifies whether a Service Location Protocol is used or not. When `true`, uses SLP. When `false`, does not use SLP.	`true` or `false`	`false`
SLPMaxWaitTime	SLPMaxWaitTime in milliseconds to wait for service response. This property is only valid when the SLPEnabled property is true.		`200`
SLPScope	Service Location Protocol (SLP) Scope is displayed as `Scope` under `SLP Options` on property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, and 5250 Printer session types.		
SLPThisScopeOnly	Session is established only to a server that supports the provided scope. This property is valid only when the SLPEnabled property is `true` and there is a SLPScope provided.	`true` or `false`	`false`
smartOrdering	Specifies whether a segment of characters with different text attributes is ordered separately. This property applies BIDI Logical VT sessions only.	`SMART_ORDERING_OFF` and `SMART_ORDERING_ON`	`SMART_ORDERING_OFF`
SSHPublicKeyAlias	Specifies the SSHPublicKeyAlias.		`mykey`
SSHPublicKeyAliasPassword	Specifies the password to read the public key information from the keystore.		
SSL	Specifies whether to use the Secure Socket Layer (SSL) feature. When `true`, enables SSL. When `false`, disables SSL.	`true` or `false`	`false`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
SSLBrowser-KeyringAdded	Specifies the SSLBrowserKeyringAdded property of the session. When `true`, adds the session to the Host On-Demand client keyring. When `false`, does not add the session to the Host On-Demand keyring.	`true` or `false`	`false`
SSLCertificate-Hash	Specifies the SSLCertificateHash.		
SSLCertificate-Name	Specifies the SSLCertificateName.		
SSLCertificate-Password	Specifies the SSLCertificatePassword.		
SSLCertificate-PromptBefore-Connect	Specifies whether the client is prompted before connecting to the server. When `true`, prompts the client. When `false`, does not prompt the client.	`true` or `false`	`false`
SSLCertificate-PromptHow-Often	Specifies how often the client is prompted.	`SESSION_SSL_CERTIFICATE_PROMPT_EACH_CONNECT`, `SESSION_SSL_-CERTIFICATE_PROMPT_FIRST_CONNECT`, `SESSION_SSL_CERTIFICATE_-PROMPT_ONLY_ONCE`	`SESSION_-SSL_CERTIFI-CATE_PROMPT_-FIRST_CONNECT`
SSLCertificate-Provided	Specifies whether the client has a certificate. The value is `true` if the client has a certificate and `false` if the client does not have a certificate.	`true` or `false`	`false`
SSLCertificate-Source	The certificate can be kept in the client's browser or dedicated security device, such as a smart card, local or network-accessed file. Displayed as `Certificate Source` on property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, 5250 Printer, and VT Display session types.	• `SSL_CERTIFICATE_IN_CSP`: For certificate in browser or security device • `SSL_CERTIFICATE_IN_URL`: For certificate in URL or file	`SESSION_SSL_-CERTIFICATE_-IN_URL`
SSLCertificate-URL	Specifies the default location of the client certificate. Displayed as the `URL or Path and Filename` in property panels. Applies to 3270 Display, 3270 Printer, 5250 Display, 5250 Print-		

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	er, and VT Display session types. The URL protocols that you can use depend on the capabilities of your browser. Most browsers support HTTP, HTTPS, FTP, and FTPS.		
SSLServer-Authentication	Specifies whether SSL server authentication is enabled.	`true` or `false`	`false`
SSLTelnetNegotiated	Specifies whether SSL will be negotiated on the Telnet connection. Set this property to `true` only if you connect to a Telnet server that supports IETF Internet-Draft `TLS-based Telnet Security`. This Internet-Draft defines the protocol for doing the SSL Handshake over a Telnet connection. Set the SSL property to `true` also.		`false`
ssoCMServer	Specifies the sso_CMServer property.	Address strings of back-end servers and applications that respond to single sign-on (SSO) queries.	
ssoEnabled	Specifies that the session is SSO enabled. When `true`, enables SSO for the session. When `false`, disables SSO.	`true` or `false`	`false`
ssoUseKerberos-Passticket	Specifies whether the SSO layer uses the client side Kerberos support to acquire a Kerberos passticket for login. When `true`, instructs the SSO layer to use the client side Kerberos support. When `false`, instructs the SSO layer to not use the client side Kerberos support.	`true` or `false`	`false`
ssoUseLocalIdentity	Specifies whether the SSO layer uses the local operating system userID in the SSO process. When `true`, instructs the client to use the local operating system user ID in the SSO process. When `false`, instructs the client not to use the local operating system user ID in the SSO process.	`true` or `false`	`false`
symmetricSwapEnabled	Specifies whether symmetric swapping is enabled. This property applies to Arabic 3270 sessions only. When `true`, enables symmet-	`true` or `false`	`true`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	ric swapping. When `false`, disables symmetric swapping.		
textOrientation	Specifies whether the test orientation is left to right or right to left. This property applies to bidirectional Sessions only.	`LEFTTORIGHT` or `RIGHTTOLEFT`	`LEFTTORIGHT`
textType	Specifies whether the textType is visual or logical. This property applies to bidirectional sessions only.	`VISUAL` or `LOGICAL`	`VISUAL`
textTypeDisp	Determines whether a session works in the logical or visual mode. This property applies BIDI VT sessions only.	`LOGICAL_DISP` and `VISUAL_DISP`	`LOGICAL_DISP`
ThaiDisplayMode	This method applies to Thai host machines only.	Integers 1 through 5	`5`

Value	Description
`1`	Non-composed mode
`2`	Composed mode
`3`	Composed mode with space alignment
`4`	Composed mode with EOF alignment
`5`	Composed mode with space and EOF alignment

timeout	Specifies the amount of time (in milliseconds) that the client waits for data. If no data is received for the specified amount of time, the session is disconnected. A value of 0 specifies that system will not time out.		`0`
timeoutNoDataAtInitialization	Specifies whether to time out if no data is received at the session initialization.	`true` or `false`	`false`
trimRectRemainAfterEdit	Specifies whether trim rec remains after a cut, copy, or paste action. When `true`, sets trim rec to remain after a cut, copy, or paste action. When `false`, does not set trim rec to remain after a cut, copy, or paste action.	`true` or `false`	`false`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
trimRectSizing-Handles	Specifies whether trim rec is sizeable or not. When `true`, trim rec is sizeable. When `false`, trim rec is not sizeable.	`true` or `false`	`true`
unicodeData-StreamEnabled	Specifies whether the session can receive Unicode data fields that are sent by a host. When `true`, the session can receive Unicode data field that is sent by a host. When `false`, the session cannot receive Unicode data field that is sent by a host.	`true` or `false`	`false`
userID	Specifies the user ID that is used in the SSH authentication process is either by public-key or password.	Valid user ID	
userPassword	Specifies the user password that is used in the SSH authentication process.	Valid user password	
useSSHPublicK-eyAuthentication	Specifies whether the SSH public key authentication is enabled. When `true`, enables the SSH public key authentication. When `false`, disables SSH public key authentication.	`true` or `false`	`false`
VT100Plus	Specifies whether VT100+ mode is enabled. In VT100+ mode, the function keys generate `ESC OP` through `ESC O[` sequences.	`true` or `false`	`false`
VTAnswerBack-Msg	A string that is returned to the remote VT server in response to an ENQ command (0x05). This string can be empty ("") or a user-defined value.		`none`
VTasciiConvert			`false`
VTAutowrap	Sets the VTAutowrap property. This property applies to VT sessions only. When `true`, enables autowrap and `false` disables autowrap.	`true` or `false`	`false`
VTBackspace	Sets the VTBackspace property. This property applies to VT sessions only. When `true`, sets normal backspace behavior and `false` deletes the character at the cursor.	`true` or `false`	`false`
VTCursor	Sets the VTCursor property. This property applies to VT sessions only. When `true`, estab-	`true` or `false`	`false`

Table 24. Advanced connection properties (continued)

Property	Description	Valid Values	Default Value
	lishes application-controlled cursor behavior and `false` establishes normal cursor behavior.		
VTID	This ID is used to determine how the emulator identifies itself to the host. This field tells the UNIX host the type of VT terminal that you want to emulate for your session.		`VT420`
VTKeypad	Sets the VTKeypad property. This property applies to VT sessions only. When `true`, sets application keypad control and `false` sets normal keypad behavior.	`true` or `false`	`false`
VTLocalEcho	Local-echo mode. This property applies to VT sessions only. If `true`, turns local echo on. If `false`, turns local echo off.	`true` or `false`	`false`
VTNewLine	New-line operation. This property applies to VT sessions only. If `true`, interprets a carriage return as CR only. If `false`, interprets carriage return as CR and LF.	`true` or `false`	`true`
VTReverse-Screen	Sets the VTReverseScreen property. This property applies to VT sessions only. When `true`, sets reverse video and `false` sets normal video.	`true` or `false`	`false`
VTTerminalType	The terminal-type required by the server to which the session will connect to. This property applies to VT sessions only.	Integers from `1` to `5`	`1`

Value	Description
1	VT220_7_BIT
2	VT220_8_BIT
3	VT100
4	VT52

SSH overview

The Secure Shell (SSH) is a set of protocols for implementing secure sessions over a non-secure network, such as a standard TCP/IP network. The Extension for Terminal-based Applications supports SSH connections to VT sessions.

In order to use SSH, you must set up SSH server software on the host.

SSH include the following features:

- Secure remote login
- Strong authentication of server and client
- Several user authentication methods
- Encrypted terminal sessions
- Secure file transfers

SSL overview

The Secure Sockets Layer (SSL) protocol creates a standard SSL connection between the client and the server. The client contacts the server and checks to make sure that the server has a valid certificate. This type of connection ensures that all data exchanged between client and server is encrypted, and is therefore not readable by a third party on the Internet.

Extension for Terminal-based Applications supports SSL connections to host sessions using Host On-Demand SSL functions. The SSL connection to a 3270 session is set up by using an open source SSL package called Stunnel.

Stunnel is a program that enables encrypting arbitrary TCP connections inside SSL and is available on both UNIX and Windows operating systems. With Stunnel you can secure non-SSL aware daemons and protocols by providing the encryption that requires no changes to the daemon code.

Using SSL to connect to host machines

IBM® Rational® Functional TesterHCL OneTest™ UI Extension for Terminal-based Applications requires a security utility such as OpenSSL or IBM® Certificate Management to produce the *.p12 file that will pass the host servers self-signed certificate credentials to the terminal to allow a secure connection. Although IBM® Certificate Management that runs on Windows®, Linux®, AIX®, or Solaris distributed platforms is not included with Rational® Functional TesterHCL OneTest™ UI, it is shipped with other IBM® products such as IBM® Personal Communications, IBM® Host On-Demand, and IBM® HTTP server. You can easily create the *.p12 file if you have access to this utility.

Before you begin

Note: Starting from 9.1.1, you can import certificates from the server to connect to the host machines securely. See Importing certificates from the server for secured connections on page 957.

To use SSL to connect to host machines, you need:

- The extracted host or server certificate in the form of an *.arm or *.der file.
- The secure port for your host connection.
- A CustomizedCAs.p12 with a password of "hod" created using IBM® Certificate Management.
- The correct settings for the terminal session.

About this task

You must use IBM® Certificate Management to create the *.p12 file. You must have access to IBM® Certificate Management tool. You must either install it or work with an existing installation:

1. Start **IBM Key Management**.
2. Click **KeyDatabase File > New**. You must change the file type to PKCS12 and name it as CustomizedCAs.p12.
3. Save the file to the folder `C:\Program Files\IBM\SDP70Shared\plugins \com.ibm.test.terminal.7.0.2v200906180724. C:\Program Files\HCL\HCLOneTest \plugins\com.ibm.test.terminal.7.0.2v200906180724.`. The terminal.jar and TerminalTester.jar must be present in this folder.

   Note: For Rational® Functional TesterHCL OneTest™ UI Extension, version 7.01, the location for the CustomizedCAs.p12 file will be `C:\Program Files\IBM\SDP70Shared\plugins \com.ibm.test.terminal.7.0.1v200709190143`

4. Type **hod** as the password.

   Note: This password is hard-coded and must be hod.

5. To add the extracted *.der or *.arm file from the host's server certificate to the CustomizedCAs.p12, click **Add**.
6. In the **Token Label** field, type a valid token label for this certificate.
7. To save the file with the certificate you just added, click **Key Database File > Save As**. Verify the password and close **IBM Key Management**.
8. Start **IBM Extension for Terminal-based ApplicationsHCL OneTest UI Extension for Terminal-based Applications**.
9. Configure the advanced settings in the Advanced Properties page of the IBM® Extension for Terminal-based Applications dialog box , click Advanced as follows:

   a. Set SecurityProtocol to `SESSION_PROTOCOL_SSL` or `SESSION_PROTOCOL_TLS`

   b. Set SSL to `true`

   c. Set SSLCertificateName to `CustomizedCAs.p12`

   d. Set SSLCertificatePassword to `hod`

   e. (optional) Set SSLCertificateProvided to `true`

f. Set SSLTelnetNegotiated to `true`.

Note: You must set SSLTelnetNegotiated to `true` only when you connect to a Telnet server that supports IETF Internet-Draft TLS-based Telnet Security. The Internet-Draft defines the protocol for performing the SSL Handshake over a Telnet connection.

10. In the terminal session under Port, type the secure port number to be used by the server connection. Typically, this is 992, but it may vary depending on the telnet configuration of your host. The secure connection must show MA*+ in the Operator Information Area at the bottom of the screen.
11. Click **OK**.

Importing certificates from the server for secured connections

Starting from Rational® Functional TesterHCL OneTest™ UI 9.1.1, you can import certificates from the server to connect to the host machines securely. You can create a `CustomizedCAs.p12 / CustomizedCAs.jks` keystore through the Extension for Terminal-based Applications that produces the *.p12/JKS file. This file passes the host server's self-signed certificate credentials to the terminal to allow a secure connection.

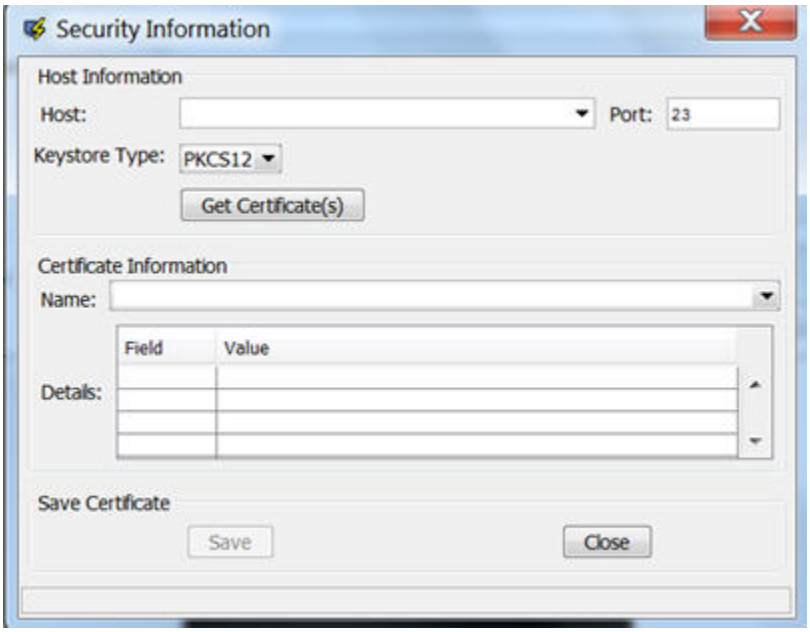
About this task

Prior to version 9.1.1, you could use SSL to connect to the host machine securely. See Using SSL to connect to host machines on page 1036.

1. Start the Extension for Terminal based Applications by clicking the Launch button in Rational® Functional TesterHCL OneTest™ UI.



2. Click **Session > Security** to open the Security Information window.



3. Select or type the Host address and Port number.
4. Select the type of keystore where the certificate can be saved. Depending on the connection protocol supported by the host, you can choose **PKCS12** or **JKS**.
5. Click the **Get Certificate(s)** button to retrieve the certificates from the host.
6. After the certificates are retrieved, click the **Save** button to save the extracted certificate to the appropriate keystore (`CustomizedCAs.p12` or `CustomizedCAs.jks`).
7. Click the Status bar to open the location where the keystore is created with the certificate. This location would be `C:\Users\<current user>\Application Data\ibm\RFT\Extension for Terminal-based Applications` on a Windows™ machine.
8. Copy the `.p12` or `.jks` file to the `<IBMIMShared\plugins>\com.ibm.test.terminal_8.5.0.vXXXX` `<HCLIMShared\plugins>\com.ibm.test.terminal_8.5.0.vXXXX` folder. This plugin folder also includes the `terminal.jar` and `TerminalTester.jar` files.
9. Close the Security Information window and restart the Extension for Terminal-based Applications.
10. Type the Host address, Port number, and terminal type information and click the **Advanced Settings** button.
11. Set the properties depending on the type of certificate.
    - For `CustomizedCAs.p12`, you must set the following properties:.

Property Name	Set the value...
SSL	*true*
SSLTelnetNegotiated	*true*

    - For `CustomizedCAs.jks`, you must set the following properties:

Property Name	Set the value...
sslUseJSSE	*true*
sslJSSETrustStore	Provide the full path of `CustomizedCAs.jsk`. For example, `C:\Program Files\IBM\IBMIMShared\plugins\com.ibm.test.terminal_8.5.0.v20170703_0428\CustomizedCAs.jksC:\Program Files\HCL\HCLIMShared\plugins\com.ibm.test.terminal_8.5.0.v20170703_0428\CustomizedCAs.jks`
sslJSSETrustStorePassword	*hodpwd*
tlsProtocolVersion	*TLSv1.2* **Note:** If the host supports an older version of the protocol, the application will fall back to the older version.
sslJSSETrustStoreType	*jks*
SSL	*true*
SSLTelnetNegotiated	*true*

Note: You must set SSLTelnetNegotiated to *true* only when you connect to a Telnet server that supports IETF Internet-Draft TLS-based Telnet Security. The Internet-Draft defines the protocol for performing the SSL Handshake over a Telnet connection.

Recognition properties

You can set the recognition property for screens, fields and characters to increase the flexibility of your script when defining a screen. Each object has a set of recognition properties, which are typically established during recording. To find an object in the application-under-test during playback, Rational® Functional TesterHCL OneTest™ UI compares the object in the application with recognition properties in the test object map. Each property of a test object has an associated recognition weight value, which is a number from 0 to 100. Rational® Functional TesterHCL OneTest™ UI uses the weight value for each recognition property to determine the importance of the property.

Table 1 describes the default recognition properties and weights for windows.

Table 25. Default recognition properties for screens

Property	Weight
Field Count	40
Non-static Field Count	40
First Field Starting Location	20
First Field Length	20
First Field Text	20
Last Field Starting Location	20
Last Field Length	20
Last Field Text	20
Text	0

Table 2 describes recognition properties and weights for fields.

Table 26. Default recognition properties for fields

Property	Weight
Starting Column	30
Starting Row	40
Length	30
Text	30

Table 3 describes recognition properties and weights for characters.

Table 27. Default recognition properties for characters

Property	Weight
Starting column	40
Starting Row	60
Position	60
Character	60

Troubleshooting issues

You can find information about the issues or problems that you might encounter while you work in the Functional Test perspective of Rational® Functional TesterHCL OneTest™ UI. You can also find the details about the causes and resolutions that you can apply to fix the issues.

The following table lists the troubleshooting issues that you might encounter in the **Functional Test** perspective.

Table 28. Troubleshooting issues

Problem	Description	Solution
If your computer is locked during the playback of a functional test script, the script fails to identify the correct control and the playback fails.	The playback fails because the inputChars() or inputKeys() methods that are used in the test script do not place the cursor at the correct text entry point to enter the text when the computer is locked or inactive.	In the test script, you can replace click() & inputChars() and click() & inputKeys() with the setText() api method.

Extending the UI Test perspective with custom code

You can use custom code to extend the default Web UI testing capabilities. You can write custom Java™ code and call the code from the test. You can also specify that results from the tests that are affected by your custom code to be included in reports.

You can find the following information:

- Creating custom Java code on page 1042
- Test execution services interfaces and classes on page 1044
- Reducing the performance impact of custom code on page 1047
- Custom code examples on page 1048

Creating custom Java™ code

Custom code uses references in the test as input and returns modified values to the test. Use the ICustomCode2 interface to create custom code and the ITestExecutionServices interface to extend test execution. These interfaces are contained in the `com.ibm.rational.test.lt.kernel.services` package.

About this task

Note: When you use the ITestExecutionServices interface in your custom code to report test results, the results for the custom code are displayed in the test log. If you log custom verification point verdicts, these are reflected in the overall schedule verdict.

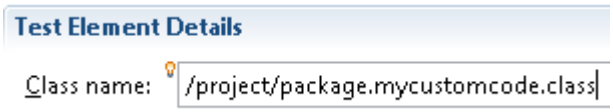
Custom code input values can be located in references or field references. You can also pass a text string as an argument to custom code. References that are used as input to custom code must be included in the same test as the custom code. In the test, the reference must precede the code that it affects. Verify that the test contains the references that are required for customized inputs to your code. For details about creating references and field references, see Creating a Reference or a Field Reference.

If your custom code uses external JAR files, you might need to change the Java™ build path. In some cases, you can avoid changing the build path manually by running the test before adding your custom code to it. The first time a test runs, classes and libraries that are required for compilation are added to the build path. For example, you can import Test and Performance Tools Platform (TPTP) classes that are required to create custom events in the test log if the test, to which you have added your custom code, has run previously. However, if the test has never been run, import errors occur because the classes are not named in the build path for the project until the test has run.

If your code uses external resources, for example, an SQL database or a product that manages customer relationships, you must configure the custom code to work on every computer on which your test runs.

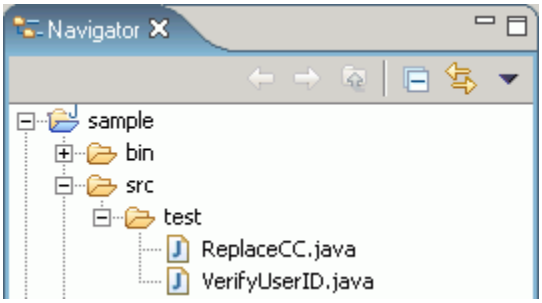
Custom code is saved in the `src` folder of the project that contains the test that calls the code. By default, custom code is located in a package named `test` in the `src` folder.

You can reuse a custom code package for tests that are located in multiple projects. The projects must be in one workspace. To reuse custom code across projects, use the project name before the custom code package. For example,



.

The following example shows the standard Navigator view of two custom code classes. (The Test Navigator does not display Java™ source files.)



When you add the `ReplaceCC.java` and `VerifyYUserID.java` custom code classes to the test and return a value to the test, **Substitute** lists these two classes.

The test package also contains the generated Java™ code for tests in the project.

You can put custom code in a different package (for example, `custom`). Separate custom code from generated code, especially if you use a source-control system.

1. Open the test, and then select either a **Launch application** or **In application** node.
2. Click **Insert**, and then select **Custom Code**.
   The **Insert** option adds the custom code above the selected node.

   Note: After you insert custom code, the **Problems** view displays an error stating that the new custom code element has no Java™ file. This error message remains until you click **View Code** or **Generate Code**, to remind you that the custom code test element is not yet associated with any Java™ code.

3. Inspect the **Class name** field, and complete one of these steps:
   - If the code to call already exists, change the class name to match its name. Click **View Code** to open the code in the Java™ editor.
   - If the code does not exist, change the class name to describe the purpose of the code. Click **Generate Code** to generate a template class for logging results and to open it in the Java™ editor. If a class with this name exists, you are warned that it will be overwritten.
4. In the **Arguments** field, click **Add**.
5. In the **Custom Code** window, select all inputs that your code requires.
   The **Custom Code** window lists all values in the test that can be used as inputs to your code (references or field references in the test that precede the code).
6. Click **OK**.
   Result
   The window closes, the selected references are added to the **Arguments** field.
7. To add text strings as inputs to your custom code, click **Text**, and then type the text string to use.
8. In the test, after your custom code, locate a value that your code returns to the test.
9. Highlight the value.
10. Right-click the highlighted value, click **Substitute**, and select the class name of your custom code.
    Result
    The custom code classes that you have added are listed. After you have made your selection, the value to be returned to the test is highlighted in orange, and the **Used by** table is updated with this information.

What to do next

Custom code is not displayed in the **Test Navigator** view. To view custom code, open the **Package Explorer** view and use the Java™ tools to identify the custom code that you added.

Test execution services interfaces and classes

You use the test execution services interfaces and classes to customize how you run tests. These interfaces and classes are located in the com.ibm.rational.test.lt.kernel package. Each interface and class is described briefly in this topic and in detail in the Javadoc information.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Interface	Description
ICustomCode2	Defines customized Java™ code for test execution services. Use this interface to create all custom code.
ITestExecutionServices	Provides information for adding custom test execution features to tests. Replaces the IKLog interface. All the methods that were available in IKLog are contained in ITestExecutionServices, along with several newly exposed objects and interfaces. This interface is the primary interface for execution services. ITestExecutionServices contains the following interfaces: • IDataArea • IARM • IDataSetController • ILoopControl • IPDLogManager • IStatisticsManager2 • ITestLogManager • ITime • ITransaction • String
IDataArea	Defines methods for storing and accessing objects in data areas. A data area is a container that holds objects. The elements of a data area are similar to program variables and are scoped to the owning container. To use objects specific to a protocol, you should use objects provided by that protocol that are stored in the protocol-specific data area.
IARM	Provides information about defining ARM (Application Response Measurement) specifications. You use this interface if your virtual users are being sampled for ARM processing.
ILoopControl	Provides control over loops in a test or schedule. For example, you can use this interface to break loops at specific points in a test. The loop that is affected is the nearest containing loop found in either the test or the schedule.
IPDLogManager	Provides logging information such as problem severity, location levels, and error messages.

Interface	Description
IStatisticsManager	Provides access to performance counters in the ICustomCode2 interface (used for defining custom code). Performance counters are stored in a hierarchy of counters. Periodically, all the counter values in the hierarchy are reported to the testing workbench and collected into test run results, where they are available for use in reports and graphs. Each counter in the hierarchy has a type (defined in class `StatType`). The operations that are available on a counter depend on the counter's type.
ITestLogManager	Logs messages and verification points to the test log. Use this interface for handling error conditions, anomalies in expected data or other abstract conditions that need to be reported to users, or for comparisons or verifications whose outcome is reported to the test log. ITestLogManager can also convey informational or status messages after the completion of a test.
ITime	Defines basic time services, such as the current system time in milliseconds (adjusted so that all systems are synchronized with the schedule controller), the time the test begins, and the elapsed time from the beginning of the test.
ITransaction	Provides support for transactions. A collection of named transactions is maintained for each virtual user. Transactions created in custom code can be started and stopped wherever custom code can be used. These transactions can span several tests. Performance counters are kept for custom code transactions and appear in reports. An example of how you could use ITransaction is to create transactions for one virtual user but not another, to help verify responses from tests.
IEngineInfo	Provides information about the testing execution engine; for example, the number of virtual users running in this engine, the number of virtual users that have completed, the local directory in which test assets are deployed, and the host name of the computer on which the engine runs.
ITestInfo	Provides information about the test that is running; for example, the test name and information about the current problem determination log level for this test.
IVirtualUserInfo	Provides information about virtual users; for example, the virtual user's name, problem determination log level, TestLog level, globally unique ID, and user group name.
IScalar	Provides methods for simple integer performance counters. It is used for counters of `SCALAR` and `STATIC` types. Use this interface to decrement and increment counters.
IStat	Defines observational performance counters. It defines the method for submitting a data point to performance counters of type `RATE`, `AVERAGE`, and `RANGE`.
IStatistics	Retrieves the performance counter tree associated with the current statistics processor. Stops the delivery of performance counters. Changes the priority of the statistics delivery thread.
IStatTree	Provides methods that can retrieve child counters, create the XML fragments that define counters, and set the description field of counters.
IText	Contains text-based performance counters. Performance counters that do not fit any of the other counter types can be created as type `TEXT`. `TEXT` counters are not assigned definitions, but they are collected in the test results.

Class	Description
`DataAreaLockException`	Throws an exception whenever an attempt is made to modify a locked DataArea key.
`OutOfScopeException`	Indicates that an object created by ITestExecutionServices has been referenced outside of its intended scope.
`TransactionException`	Throws an exception when a transaction is misused. The following conditions lead to a `TransactionException` exception: attempting to start a transaction that has already been started, attempting to stop a transaction that has not been started, and getting the start time or the elapsed time of a transaction that has not been started. Any operation (except abort()) on a transaction that has been aborted will throw a `TransactionException` exception.
`StatType`	Provides a list of valid performance counter types. The performance counter types are: `AVERAGE, iAVERAGE, iRANGE, iRATE, iSCALAR, iSTATIC, iSTRUCTURE, iTEXT, RANGE, RATE, SCALAR, STATIC, STRUCTURE`, and `TEXT`.

Reducing the performance impact of custom code

If custom code runs inside a page, it can affect that page's response time.

HTTP pages are containers of HTTP requests. On a given HTTP page, requests run in parallel across all of the connections between the agent computer and the system under test.

Page response time is the interval between *page start* and *page end*, which are defined as follows: Page start is the first timestamp associated with the client-server interaction. This interaction is either the first byte sent or the first connect of the first HTTP request. Page end is the last timestamp associated with the client-server interaction. This interaction is the last byte received of the last HTTP request to complete. Because of parallelism, the last HTTP request to complete might not be the last one listed for the page.

Typically, you should not insert custom code inside a page. While custom code that runs for only a few milliseconds should have little effect on page response time, the best practice is to place custom code outside a page. Custom code placed outside a page has no effect on page response time, and its execution time can overlap with think time delays.

Do not use custom code for data correlation if you can instead use the data correlation features built into the product. The built-in data correlation code takes advantage of requests running in parallel, whereas custom code actions do not begin until all earlier actions are completed.

You might need to place custom code inside a page to correlate a string from the response of a request inside that page to another request inside the same page. Even in this case, if you split the page into two pages, you can use the built-in data correlation features instead of custom code.

If you still want to run tests with custom code inside HTTP pages, use the Page Element report to evaluate performance. The Page Element report shows the response time and throughput for individual HTTP requests. Custom code does not affect the response time measurement of individual HTTP requests.

Custom code examples

Custom code enables you to perform such tasks as managing loops, retrieving virtual user information, running external programs from tests, and customizing data correlation.

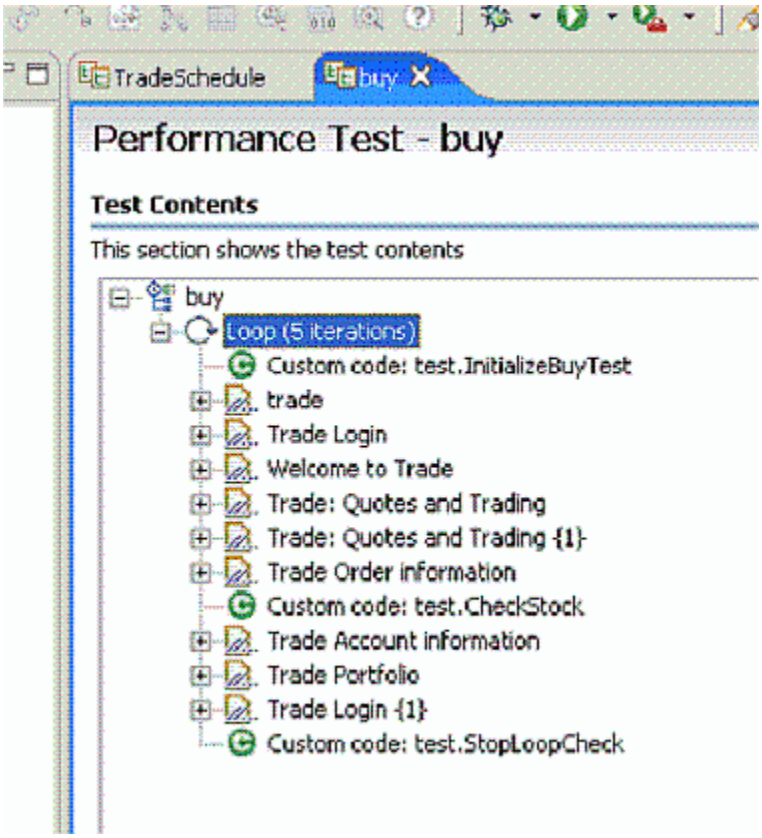
Controlling loops

This example demonstrates extending test execution by using custom code to control loops. It provides sample code that shows how you can manipulate the behavior of loops within a test to better analyze and verify test results.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

This example uses a recording of a stock purchase transaction using the Trade application. The concepts shown here can be used in tests of other applications.

The test begins with a recording of a stock purchase transaction, using dataset substitution for the login IDs. The pages are wrapped in a five-iteration loop, as shown in the following figure:



Notice that among the various pages of the test, three items of custom code exist (indicated by the green circles with "C"s in them). This example explores these items of custom code.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

The first piece of custom code, `InitializeBuyTest`, is mentioned here:

```
package customcode;

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 unknown
 */
public class InitializeBuyTest implements
  com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

 /**
  * Instances of this will be created using the no-arg constructor.
  */
 public InitializeBuyTest() {
```

```
}

/**
 * For description of ICustomCode2 and ITestExecutionServices interfaces,
 * see the Javadoc.. */
public String exec(ITestExecutionServices tes, String[] args) {
 // Get the test's data area and set a flag indicating that nothing
 // has failed yet. This flag will be used later to break out
 // of the schedule loop as soon as a failure is encountered.
 IDataArea dataArea = tes.findDataArea(IDataArea.TEST);
 dataArea.put("failedYet", "false");

 // Get the virtual users's data area
 IDataArea vda = tes.findDataArea(IDataArea.VIRTUALUSER);

 // Randomly select a stock to purchase from the set of s:0 to s:499.
    IVirtualUserInfo vuInfo = (IVirtualUserInfo) vda.get(IVirtualUserInfo.KEY);
    Random rand = vuInfo.getRandom();
 String stock = "s:" + Integer.toString(rand.nextInt(499));

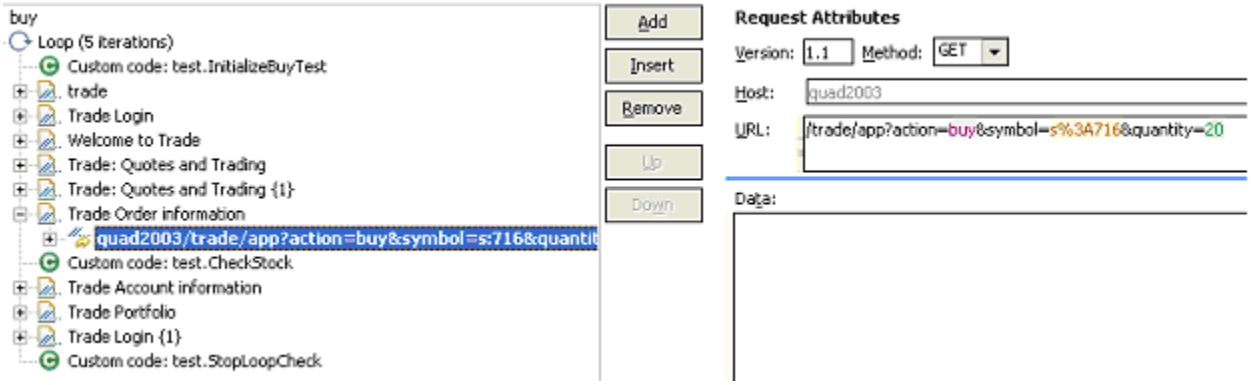
 // Persist the name of the stock in the virtual user's data area.
 vda.put("myStock", stock);

 return stock;
}
```

This custom code is located in the method `exec()`.

First, the data area for the test is acquired to store a flag value, in this case a string of text, to be used later to stop the test loop when an error is discovered. Data stored in this way can be persisted across tests.

Then a randomly generated stock string is created. The value is stored as the variable *stock*, and is passed back as the return value for the method. This return value is used as a substitute in a request later, as shown in the following figure:



The highlighted item uses a substitution (`s%3A716`), which is the value returned by the `InitializeBuyTest` custom code item. We are using custom code to drive the direction of our test.

The next lines of code in `InitializeBuyTest` use the Virtual User data area to store the name of the stock for later reference. Again, data stored in this way can persist across tests.

The second piece of custom code is called `CheckStock`. Its contents are as follows (listing only the `exec()` method this time):

```
public String exec(ITestExecutionServices tes, String[] args) {

  // Get the actual and requested stock purchased.
  String actualStock = args[0].replaceAll("<B>", "");
   actualStock = actualStock.substring(0, actualStock.indexOf("<"));
  String requestedStock = args[1];

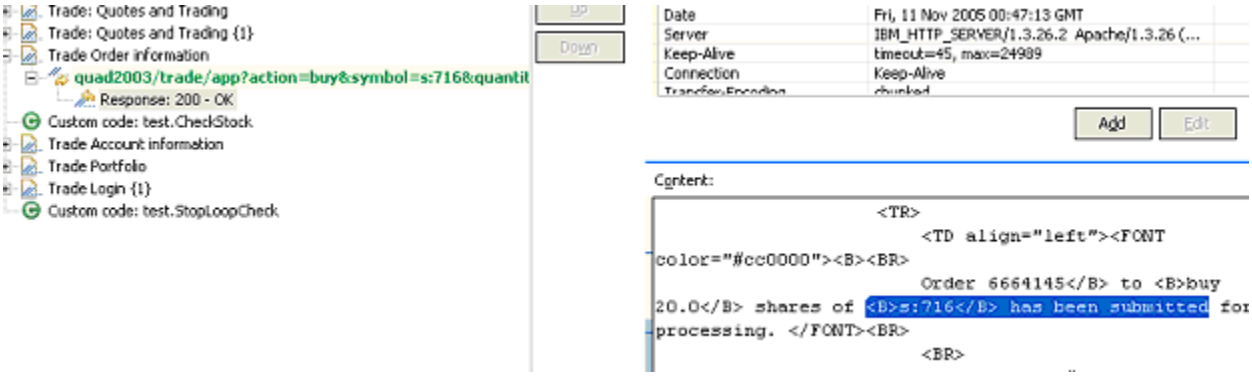
    // Set the log level to ALL.
   IDataArea dataArea = tes.findDataArea(IDataArea.TEST);
   ITestInfo testInfo = (ITestInfo)dataArea.get(ITestInfo.KEY);
   testInfo.setTestLogLevel(ITestLogManager.ALL);

   // If the log level is set to ALL, report the actual and requested stock
  // purchased.
   ITestLogManager testLogManager = tes.getTestLogManager();
  if (testLogManager.wouldReport(ITestLogManager.ALL)) {
    testLogManager.reportMessage("Actual stock purchased: "
      + actualStock + ". Requested stock: " + requestedStock
     + ".");
   }

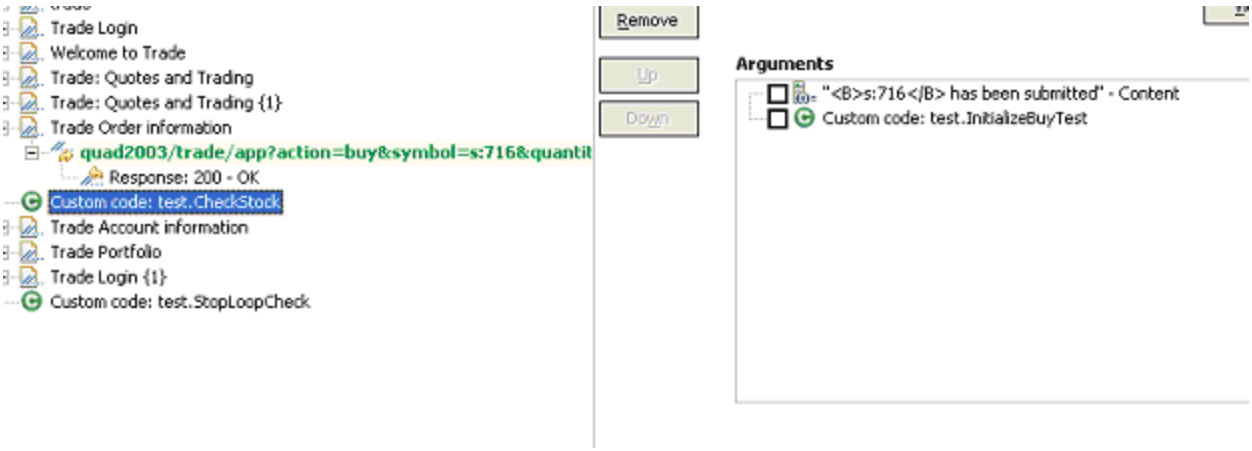
  // If the actual and requested stock don't match, submit a FAIL verdict.
   if (testLogManager.wouldReport(ITestLogManager.ALL)) {
   if (!actualStock.equalsIgnoreCase(requestedStock)) {
     testLogManager.reportVerdict(
      "Actual and requested purchase stock do not match.",
       VerdictEvent.VERDICT_FAIL);

     // Use the test's data area to record the fact that an error has
    // occurred.
     dataArea.put("failedYet", "true");
    }
  }
    return null;
 }
```

This code begins by extracting two arguments that have been passed to the code. A part of the response in the original recording is highlighted and used as a reference, as shown in the following figure.



Some string manipulation is needed to acquire the text of interest; in this case, the name of the stock that was actually purchased. This newly created reference is then passed into `CheckStock` as an argument, as shown in the following figure:



Note that the return value of `InitializeBuyTest` is passed in as an argument as well.

The `CheckStock` custom code item uses these values to verify that the randomly chosen stock generated by `InitializeBuyTest` is actually purchased during the execution of the test.

`CheckStock` then sets the test log level, reports the actual and requested stock purchase, and raises a FAIL verdict if they do not match. `CheckStock` also stores a `true` value associated with the tag `failedYet` in the test's data area.

The third piece of custom code (`exec()` method only) is mentioned here:

```
public String exec(ITestExecutionServices tes, String[] args) {

  // Get the test log manager.
  ITestLogManager testLogManager = tes.getTestLogManager();

  // Get the test's data area and get a flag indicating to
  // see if anything has failed yet. If so, stop the loop.
  IDataArea dataArea = tes.findDataArea(IDataArea.TEST);
  String failedYet = (String) dataArea.get("failedYet");

  // Break out of the loop if an error has been encountered.
```

```
  if (failedYet.equalsIgnoreCase("true")) {
   tes.getLoopControl().breakLoop();

   if (testLogManager.wouldReport(ITestLogManager.ALL)) {
    testLogManager.reportMessage("Loop stopped.");
   }
  }

  return null;
 }
```

This code uses the test's data area to determine the user-defined value associated with the tag `failedYet`. If `failedYet` is `true`, `StopLoopCheck` breaks out of the test loop.

Retrieving the IP address of a virtual user

This example shows how to retrieve the local IP address of a virtual user. Retrieving IP addresses is particularly useful when virtual users are using IP aliases.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

The following custom code retrieves the IP address that was assigned to a virtual user:

```
import java.net.InetAddress;
import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;
import com.ibm.rational.test.lt.kernel.services.IVirtualUserInfo;

public String exec(ITestExecutionServices tes, String[] args) {
 IVirtualUserInfo vui = (IVirtualUserInfo)
 tes.findDataArea(IDataArea.VIRTUALUSER).get(IVirtualUserInfo.KEY);
 ITestLogManager tlm = tes.getTestLogManager();

 if (vui != null) {
  String localAddr = null;
  InetAddress ipAddr = vui.getIPAddress();
  if (ipAddr != null)
   localAddr = ipAddr.toString();
  tlm.reportMessage("IPAlias address is " + (localAddr != null ? localAddr : "not set"));
  return localAddr;
 }
else
  return ("Virtual User Info not found");
}
```

Note:

- IP aliasing must be enabled. If not, `vui.getIPAddress()` returns null.
- IP aliases must be configured at the remote location.

- The **Log Level** must be set to a value granular enough to include the IP address, so that the `tlm.reportMessage()` method can retrieve it. If you insert custom code at the page level, keep **Log Level** at the default value, **Primary Test Actions**. If you insert custom code at the request level, set **Log Level** to **Secondary Test Actions**, a more granular value.

Printing input arguments to a file

The PrintArgs class prints its input arguments to the file C:\arguments.out. This class could be used, for example, to print a response returned by the server.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.io.*;

/**
 * The PrintArgs class prints its input arguments to the file
 * C:\arguments.out.  This example could be used to print a response
 * returned by the server.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public PrintArgs() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        try {
            FileWriter outFile = new FileWriter("C:\\arguments.out");
            for (int i = 0; i < args.length; i++)
                outFile.write("Argument " + i + " is: " + args[i] + "\n");
            outFile.close();
        } catch (IOException e) {
            tes.getTestLogManager().reportMessage(
                                        "Unable to write to C:\\arguments.out");
        }
        return null;
    }
}
```

Counting the number of times that code is executed

The CountAllIterations class counts the number of times code is executed by all virtual users. The CountUserIterations class counts the number of times code is executed by an individual virtual user.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

The CountAllIterations class counts the number of times it is executed by all virtual users running in a particular JVM and returns this count as a string.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * The CountAllIterations class counts the number of times it is executed
 * by all virtual users running in a particular JVM and returns this count
 * as a string.  If all virtual users on an agent are running in the same
 * JVM (as would typically be the case), this class will count the number of
 * times it is run on the agent.
 */

/**
 * @author IBM Custom Code Samples
 */

public class CountAllIterations implements
        com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {
    private static int numJVMLoops = 0;

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public CountAllIterations() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        return Integer.toString(++numJVMLoops);
    }
}
```

Exemple

The CountUserIterations class counts the number of times code is executed by an individual virtual user.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.IDataArea;

/**
 * The CountUserIterations class counts the number of times it is executed
 * by an individual virtual user and returns this count as a string.
```

```
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public CountUserIterations() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        IDataArea userDataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
        final String KEY = "NumberIterationsPerUser";

        Number numPerUser = (Number)userDataArea.get(KEY);
        if (numPerUser == null) {
            numPerUser = new Number();
            userDataArea.put(KEY, numPerUser);
        }

        numPerUser.value++;
        return Integer.toString(numPerUser.value);
    }

    private class Number {
        public int value = 0;
    }
}
```

Setting and clearing cookies for a virtual user

The SetCookieFixedValue class sets a Cookie for a virtual user, and the ClearCookies class clears all cookies for a virtual user.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

The SetCookieFixedValue class sets a Cookie, defined in the newCookie variable, for a virtual user just as if the server had returned a Set-Cookie.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.execution.http.cookie.IHTTPVirtualUserInfo;
import com.ibm.rational.test.lt.kernel.IDataArea;

import java.text.ParseException;

/**
```

```
 * The SetCookieFixedValue class sets a Cookie, defined in the newCookie
 * variable, for a virtual user just as if the server had returned a Set-Cookie.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public SetCookieFixedValue() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String newCookie = "MyCookie=CookieValue;path=/;domain=.ibm.com";
        IDataArea dataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
        IHTTPVirtualUserInfo httpInfo =
                (IHTTPVirtualUserInfo)dataArea.get(IHTTPVirtualUserInfo.KEY);

        try {
            httpInfo.getCookieCache().setCookie(newCookie);
        } catch (ParseException e) {
            tes.getTestLogManager().reportMessage("Unable to parse Cookie " +
                                                  newCookie);
        }

        return null;
    }
}
```

The ClearCookies class clears all Cookies for a virtual user.

```
package customcode;

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

/**
 * The ClearCookies class clears all Cookies for a virtual user.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ClearCookies() {
    }
```

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

Determining where a test is running

The ComputerSpecific class determines where a test is running

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * The ComputerSpecific class determined the hostname on which the test is
 * running, prints the hostname and IP address as a message in the test log,
 * and returns different strings based on the hostname.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ComputerSpecific() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String hostName = "Unknown";
        String hostAddress = "Unknown";

        try {
            hostName = InetAddress.getLocalHost().getHostName();
            hostAddress = InetAddress.getLocalHost().getHostAddress();
        } catch (UnknownHostException e) {
            tes.getTestLogManager().reportMessage(
                                    "Not able to obtain host information");
            return null;
        }
        tes.getTestLogManager().reportMessage("The hostname is " + hostName +
                                            "; IP address is " + hostAddress);
```

```
        if (hostName.equals("host-1234"))
            return "Special";
        else
            return "Normal";
    }
}
```

Storing and retrieving variable values

You can use the getValue() and setValue() methods to store and retrieve values in variables. Depending on the storage location that you specify, variables can be shared among tests, or stored locally in the current test.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

You can use the getValue() and setValue() methods to store multiple values in variables in one custom code call. You can then create substitutions from variables instead of from multiple custom code elements.

For example, assume that a response contains three values: id, book title, and price. You can read all three values from the response, and then use custom code to set the variables *id*, *title*, and *price*. You can then substitute the values from the three variables as needed in the test, instead of having to write custom code for each variable.

Note: The storage location passed to the method must match the storage location used when declaring the variable.

```
package customcode;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
     * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
     * see the 'Extending test execution with custom code' help topic.
     */

/**
 * @author IBM Custom Code Samples
 */

    public String exec(ITestExecutionServices tes, String[] args) {

        tes.getValue("myVar", tes.STORAGE_USER);  // This retrieves a value from a test for the variable
 called myVar. The storage area is shared between tests.
        tes.getValue("myLocalVar", tes.STORAGE_LOCAL);  // This variable is stored locally, per test.

        tes.setValue("myVar", tes.STORAGE_USER, "myNewValue");  // Change the value of the variable
 myVar, which is shared between tests, to myNewValue.
        tes.setValue("myLocalVar", tes.STORAGE_LOCAL, "myLocalNewVar");  // Change the value of the
 local variable to myLocalNewVar.
```

```
        return null;
    }
```

Extracting a string or token from its input argument

The ParseResponse class extracts a string from its input argument. The ExtractToken class extracts a particular token (string) from its input argument. Both classes can be useful for handling certain types of dynamic data correlation.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

The ParseResponse class extracts a string from its input argument, using a regular expression for pattern matching.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.util.regex.*;

/**
 * The ParseResponse class demonstrates using Custom Code to extract a
 * string from its input argument using a regular expression for pattern
 * matching.
 *
 * In this sample, the args[0] input string is assumed to be the full
response  from a previous request.  This response contains the day's
headlines in a format such as:
 *
 *   <a class=f href=r/d2>In the News</a><small class=m> <span id=nw>
 *   </span></small></h2>
 *   <div class=ct>
 *   &#149; <a href=s/213231>Cooler weather moving into eastern
U.S.</a>  *   <br>&#149; <a href=s/262502>Digital camera shipments
up</a><br>  *
 * Given the above response, the extracted string would be:
 *        Cooler weather moving into eastern U.S.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ParseResponse() {}

    public String exec(ITestExecutionServices tes, String[] args) {
        String HeadlineStr = "No Headline Available";
        String RegExpStr = ".*In the News[^;]*;[^;]*;[^;]*;<a
```

```
href=([^>]*)>([^<]*)<";         Pattern pattern =
Pattern.compile(RegExpStr, Pattern.DOTALL);         Matcher matcher =
pattern.matcher(args[0]);
        if (matcher.lookingAt())
            HeadlineStr = matcher.group(2);
        else
            tes.getTestLogManager().reportMessage("Input does not match
pattern.");
        return HeadlineStr;
    }
```

The ExtractToken class extracts a particular string from its input argument.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * The ExtractToken class demonstrates using Custom Code to extract a particular
 * token (string) from its input argument.  This can be useful for handling
 * certain types of dynamic data correlation.
 *
 * In this sample, the args[0] input string is assumed to be comma-delimited
 * and the token of interest is the next-to-last token.  For example, if
 * args[0] is:
 *    javascript:parent.selectItem('1010','[Negative]1010','1010','','IBM',
 *         '30181','Rational','1','null','1','1','6fd8e261','RPT')
 * the class will return the string 6fd8e261.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    public ExtractToken() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String ArgStr;
        String NextToLastStr;
        String[] Tokens = args[0].split(",");

        if (Tokens.length> 2) {
            ArgStr = Tokens[Tokens.length - 2];        // Extract next-to-last token

            // Remove enclosing ''
            NextToLastStr = ArgStr.substring(1, ArgStr.length() - 1);
        } else {
            tes.getTestLogManager().reportMessage("Could not extract value");
            NextToLastStr = null;
        }
        return NextToLastStr;
    }
}
```

Retrieving the maximum JVM heap size

The JVM_Info class retrieves the maximum heap size of the JVM.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.net.*;

/**
 * The JVM_Info class retrieves the maximum heap size of the JVM.
 * It writes a message in the test log with the hostname where the
 * JVM is running and the JVM's maximum heap size in megabytes.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    public JVM_Info() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        Runtime rt = Runtime.getRuntime();
        long maxMB = rt.maxMemory()/(1024*1024); // maxMemory() size is in bytes
        String hostName = "Unknown";

        try {
            hostName = InetAddress.getLocalHost().getHostName();
        } catch (UnknownHostException e1) {
            tes.getTestLogManager().reportMessage("Can't get hostname");
            return null;
        }

        tes.getTestLogManager().reportMessage("JVM maximum heap size for host "
                                        + hostName + " is " + maxMB + " MB");
        return null;
     }
}
```

Running an external program from a test

The ExecTest class runs a program, defined in the execName variable, on the system where the test is running.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

Exemple

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;
import org.eclipse.hyades.test.common.event.VerdictEvent;

import java.io.IOException;

/**
 * The ExecTest class runs a program, defined in the execName variable,
 * on the system where the test is running.
 * The test verdict is set to PASS if the program return code is 0.
 * The test verdict is set to FAIL if the program doesn't execute or
 * if the program return code is non-zero
 * In this sample, the program is perl.exe.
 */

/**
 * @author IBM Custom Code Samples
 */

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

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ExecTest() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        ITestLogManager logger = tes.getTestLogManager();
        int rtnval = 1;
        Process p = null;
        String execName = "C:/Windows/System32/perl.exe C:/Perl/true.pl";

        Runtime rt = Runtime.getRuntime();
        // Execute test
        try {
            p = rt.exec(execName);
        } catch (IOException e) {
            logger.reportMessage("Unable to run = " + execName);
            logger.reportVerdict("Execution of " + execName + " failed",
                                                VerdictEvent.VERDICT_FAIL);
            return null;
        }

        // Wait for the test to complete
        try {
            rtnval = p.waitFor();
            logger.reportMessage("Process return value is " +
                                                String.valueOf(rtnval));
        } catch (InterruptedException e1) {
            logger.reportMessage("Unable to wait for " + execName);
            logger.reportVerdict("WaitFor on " + execName + " failed",
```

```
                                                VerdictEvent.VERDICT_FAIL);
            return null;
        }

        // Check the test return code and set the test verdict appropriately
        if (rtnval != 0)
        {
            logger.reportVerdict("Execution failed", VerdictEvent.VERDICT_FAIL);
        } else {
            logger.reportVerdict("Execution passed", VerdictEvent.VERDICT_PASS);
        }

        return null;
    }
}
```

Adding custom counters to reports

When you want to monitor the specific requirement, you can add custom counters to performance report by using the custom code. After running tests, the results from the custom counters are automatically aggregated in the same way that the default performance testing counters.

Starting from 10.1.0, you can view and monitor the counter information generated by the custom code on a graph when the custom code starts in the test run.

After running tests, you can view the custom counter in the report. You can also view the custom counter information on a different page by creating a custom report. For more information about customizing the report, see related links.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

You can add the following custom code in your test to create a custom counter in a report.

```
package test;

import org.eclipse.hyades.test.common.event.VerdictEvent;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.stats.CountAggregationLevel;
import com.ibm.rational.test.lt.kernel.services.stats.CounterUnits;
import com.ibm.rational.test.lt.kernel.services.stats.ICounterFolder;
import com.ibm.rational.test.lt.kernel.services.stats.ICounterRegistry;
import com.ibm.rational.test.lt.kernel.services.stats.IStatisticsManager2;
import com.ibm.rational.test.lt.kernel.services.stats.IValueCounter;
import com.ibm.rational.test.lt.kernel.services.stats.ValueAggregationLevel;

import database.DatabaseAccess;
import database.TransactionResult;

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

 private static boolean registerDone;

 /**
```

```
 * This method declares the counters that will be produced during execution.
 * Declaring counters is optional, but it allows to customize some of their
 * attributes, such as the label and unit, and what level of statistical information
 * will be available in reports.
 */
private static synchronized void registerCounters(ICounterRegistry registry) {
 if (registerDone) return;
 registry.path("Database", "Transaction", "Attempts")
  .count()
  .aggregationLevel(CountAggregationLevel.RATE_RANGE)
  .label("Started Transactions")
  .unit("transactions")
  .register();

 registry.path("Database", "Transaction", "Commits")
  .verificationPoint()
  .label("Transaction Commits VP")
  .register();

 registry.path("Database", "Transaction", "Response Time", "Network")
  .value()
  .aggregationLevel(ValueAggregationLevel.RANGE)
  .unit(CounterUnits.MILLISECONDS)
  .register();

 registry.path("Database", "Transaction", "Response Time", "Commit")
  .value()
  .aggregationLevel(ValueAggregationLevel.DISTRIBUTION)
  .unit(CounterUnits.MILLISECONDS)
  .register();

 registry.path("Database", "Error")
  .text()
  .label("Database Error Message")
  .register();
 registerDone = true;
}

private DatabaseAccess database = DatabaseAccess.INSTANCE;

/**
 * This custom code adds a record in database. It produces a couple of counters,
 * such as the database transaction attempts, successes/failures, and response time.
 */
public String exec(ITestExecutionServices tes, String[] args) {
 String product = args.length> 0 ? args[0] : "Default";
 IStatisticsManager2 mgr = tes.getStatisticsManager2();
 registerCounters(mgr.registry());

 database.startTransaction();
 mgr.getCountCounter("Database", "Transaction", "Attempts").increment();

 database.executeQuery("INSERT INTO TABLE Purchases VALUES('" + product + "', 1000)");
 TransactionResult result = database.commit();

 mgr.getVerificationPointCounter("Database", "Transaction", "Commits")
  .increment(result.isSuccess() ? VerdictEvent.VERDICT_PASS : VerdictEvent.VERDICT_FAIL);
```

```
  if (!result.isSuccess()) {
   mgr.getTextCounter("Database", "Error").addMeasurement(result.getErrorMessage());
  }

  ICounterFolder times = mgr.getFolder("Database", "Transaction", "Response Time");
  times.getValueCounter("Network").addMeasurement(result.getNetworkTime());
  times.getValueCounter("Commit").addMeasurement(result.getCommitTime());

  IValueCounter value = tes.getStatisticsManager2().getValueCounter("MyStats", "Value");
  value.addMeasurement(System.nanoTime() % 2000);

  return null;
 }

}
```

Related information

Creating custom Java code on page 1042

Creating custom reports on page 1479

Using transactions and statistics

You can use custom code to start transactions, gather additional statistics during a transaction, and stop a transaction.

The Javadoc for the test execution services interfaces and classes can be accessed from the product by clicking **Help > Help Contents > Rational Functional Tester API Reference > HCL OneTest UI API Reference**.

The following code shows how to start a transaction. Transactions that are generated by test execution services automatically create and manage statistics.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITransaction;

/**
 * @author IBM Custom Code Samples
 */
public class BeginTransaction implements
  com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

 /**
  * Instances of this will be created using the no-arg constructor.
  */
 public BeginTransaction() {
 }

 /**
  * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
  * see the 'Test execution services interfaces and classes' help topic.
  */
```

```
 public String exec(ITestExecutionServices tes, String[] args) {
  // the name of the transaction could have been passed in via data correlation mechanism.
  ITransaction foo = tes.getTransaction("foo");
  foo.start();
  return null;
 }
}
```

The following code shows how to gather additional statistics during a transaction.

```
package customcode;

import com.ibm.rational.test.lt.kernel.ITime;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.statistics.IScalar;
import com.ibm.rational.test.lt.kernel.statistics.IStat;
import com.ibm.rational.test.lt.kernel.statistics.IStatTree;
import com.ibm.rational.test.lt.kernel.statistics.impl.StatType;

/**
 * @author IBM Custom Code Samples
 */
public class BodyTransaction implements
  com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

 /**
  * Instances of this will be created using the no-arg constructor.
  */
 public BodyTransaction() {
 }

 /**
  * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
  * see the 'Test execution services interfaces and classes' help topic.
  */
 public String exec(ITestExecutionServices tes, String[] args) {
  IStatTree tranStat;
  IStatTree timeStat;
  IStatTree countStat;

  IStat timeDataStat = null;  // counter for the time RANGE
  IScalar countDataStat = null; // counter for the count SCALAR

  ITime timer = tes.getTime();

        IStatTree rootStat = tes.getStatisticsManager().getStatTree();
        if (rootStat != null) {
         // these counters set up the hierarchy
         tranStat = rootStat.getStat("Transactions", StatType.STRUCTURE);
         timeStat = tranStat.getStat("Body Time", StatType.STRUCTURE);
         countStat = tranStat.getStat("Bocy Count", StatType.STRUCTURE);

         // the name of the counters could have been passed in via data correlation mechanism
         timeDataStat = (IStat) timeStat.getStat("foo", StatType.RANGE);
         countDataStat = (IScalar) countStat.getStat("foo", StatType.SCALAR);
        }
```

```
        // get the start time
        long startTime = timer.timeInTest();

        // do the work
        // whatever that work might be

        // get the end time
        long endTime = timer.timeInTest();

        // update timeDataStat with the elapsed time
        if (timeDataStat != null)
         timeDataStat.submitDataPoint(endTime - startTime);

        // update the countDataStat
        if (countDataStat != null)
         countDataStat.increment();

  return null;
 }

}
```

The following code shows how to stop a transaction.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITransaction;

/**
 * @author IBM Custom Code Samples
 */
public class EndTransaction implements
  com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

 /**
  * Instances of this will be created using the no-arg constructor.
  */
 public EndTransaction() {
 }

 /**
  * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
  * see the 'Test execution services interfaces and classes' help topic.
  */
 public String exec(ITestExecutionServices tes, String[] args) {
  // the name of the transaction could have been passed in via data correlation mechanism.
  ITransaction foo = tes.getTransaction("foo");
  foo.stop();
  return null;
 }

}
```

Reporting custom verification point failures

You can use custom code to report a custom verification point failure.

The Javadoc for the test execution services interfaces and classes can be accessed from Java API reference on page 1916.

The following code shows how to report a custom verification point failure.

```
package customcode;

import org.eclipse.hyades.test.common.event.VerdictEvent;
import org.eclipse.hyades.test.common.runner.model.util.Verdict;

import com.ibm.rational.test.lt.execution.core.IVerificationPoint;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author IBM Custom Code Samples
 */
public class Class implements
  com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

 /**
  * Instances of this will be created using the no-arg constructor.
  */
 public Class() {
 }

 /**
  * For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select 'Help Contents' in the
  * Help menu and select 'Extending
                    Rational® Performance
                    TesterHCL OneTest™ Performance functionality' -> 'Extending test execution with
 custom code'
  */
 public String exec(ITestExecutionServices tes, String[] args) {
  tes.getTestLogManager().reportVerificationPoint("CustomVP", VerdictEvent.VERDICT_FAIL);
  return null;
 }

}
```

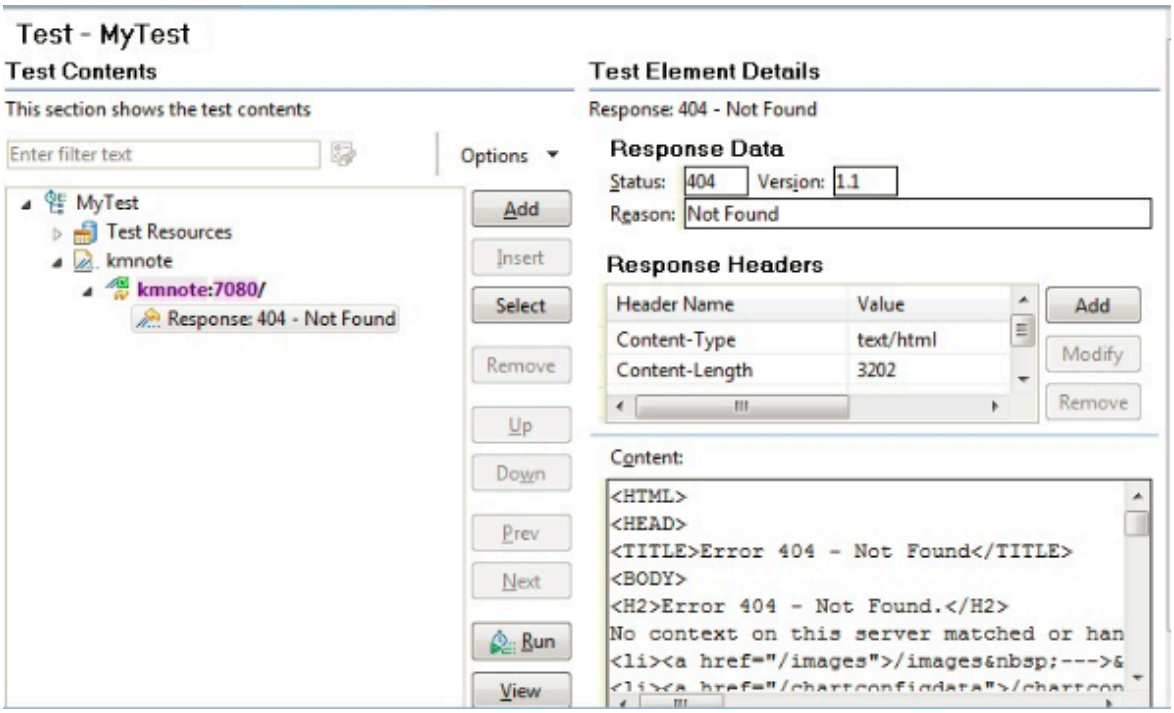
Debugging custom code

This example demonstrates debugging custom code by adding a breakpoint. It provides sample code to add a breakpoint. This way of debugging custom code is applicable only for a schedule.

1. Start IBM® Rational® Performance TesterHCL OneTest™ Performance and create a performance test project `MyProject`.
2. Create an HTTP test, **MyTest**, by recording a visit to `http://<hostname>:7080/`.

   Note: Before accessing the URL, ensure that Rational® Performance TesterHCL OneTest™ Performance is running. The URL returns an HTTP 404 error, which is expected.

   Result



3. Expand the first request and click the response element.
4. In the Test Element Details section, right-click in the **Content** field and click **Create Field Reference**.
5. Type the reference name and click **OK**.
6. Click the first page, and then click **Add > Custom Code**.
7. In the **Arguments** section of Test Element Details, click **Add**.
8. Expand the data source for the search results page, select the reference name that you created in step 5, and click **Select**.
9. Click **Generate Code**.

   Result

   A new tab with the generated code is displayed.
10. Insert the following the code into the `exec()` method:

```
ITestLogManager history = tes.getTestLogManager();
if (args.length> 0) {
    if (args[0].indexOf("Invester Relations") != -1) {
        history.reportMessage("First page failed.  Bail loop!");
        tes.getLoopControl().continueLoop();
    }
}
```

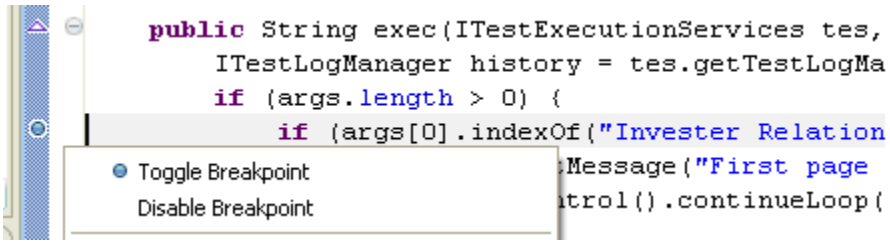
Important:

- Fix the double quotation marks, if any, so they are straight and the compiler no longer gives warning.
- To resolve complier warnings related to importing a class, press **Ctrl + Shift + O**.

The code will look like this:

```
public String exec(ITestExecutionServices tes, String[] args) {
    ITestLogManager history = tes.getTestLogManager();
    if (args.length > 0) {
        if (args[0].indexOf("Invester Relations") != -1) {
            history.reportMessage("First page failed.  Bail loop!");
            tes.getLoopControl().continueLoop();
        }
    }
    return null;
}
```

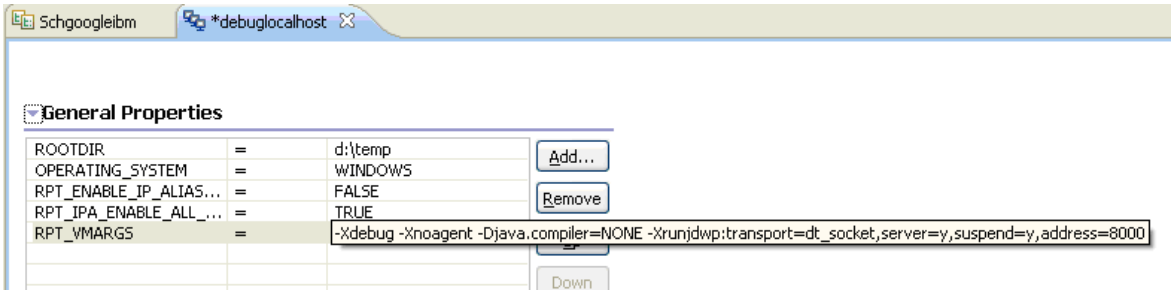
11. To set a breakpoint, click anywhere on the `args[0].indexOf` line. Move the pointer to the left-most portion of the text editor window and double-click with the pointer horizontally on the same line. A blue button is displayed in this left-most portion of the window indicating the breakpoint is set.



12. Save the custom code and then the test.
13. Create a new schedule, `Schtest`.
    a. In `Schtest`, set the number of users to run to 1.
    b. Click **User Group 1** and click **Add > Test**. Select the `MyTest` test and click **OK**.
    c. Click **User Group 1** and click the **Run this group on the following locations** button.
    d. Click **Add > Add New**.
    e. In the **New Location** window, type the following information:
        i. In **Host name**, type `localhost`.
        ii. In **Name**, type `debuglocation`.
        iii. In **Deployment directory**, type `C:\mydeploy`.
        iv. Click **Finish**.
    f. Save the schedule.
14. In the Test Navigator, right-click **debuglocation** and click **Open**.
15. Click the **General Properties** tab and click **Add**.
16. In the **Property name** field, type `RPT_VMARGS` and in the **Property value** field, add the following values each separated by a space.

```
-Xdebug
-Xnoagent
```

```
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
```



17. Save the location.
18. Attach the debugger to the schedule execution process.

    a. Run the schedule.
    Because the schedule is using **debuglocation**, it will pause at the beginning to let you attach the debugger to the execute process.

    b. Click **Window > Open Perspective > Other > Debug**.

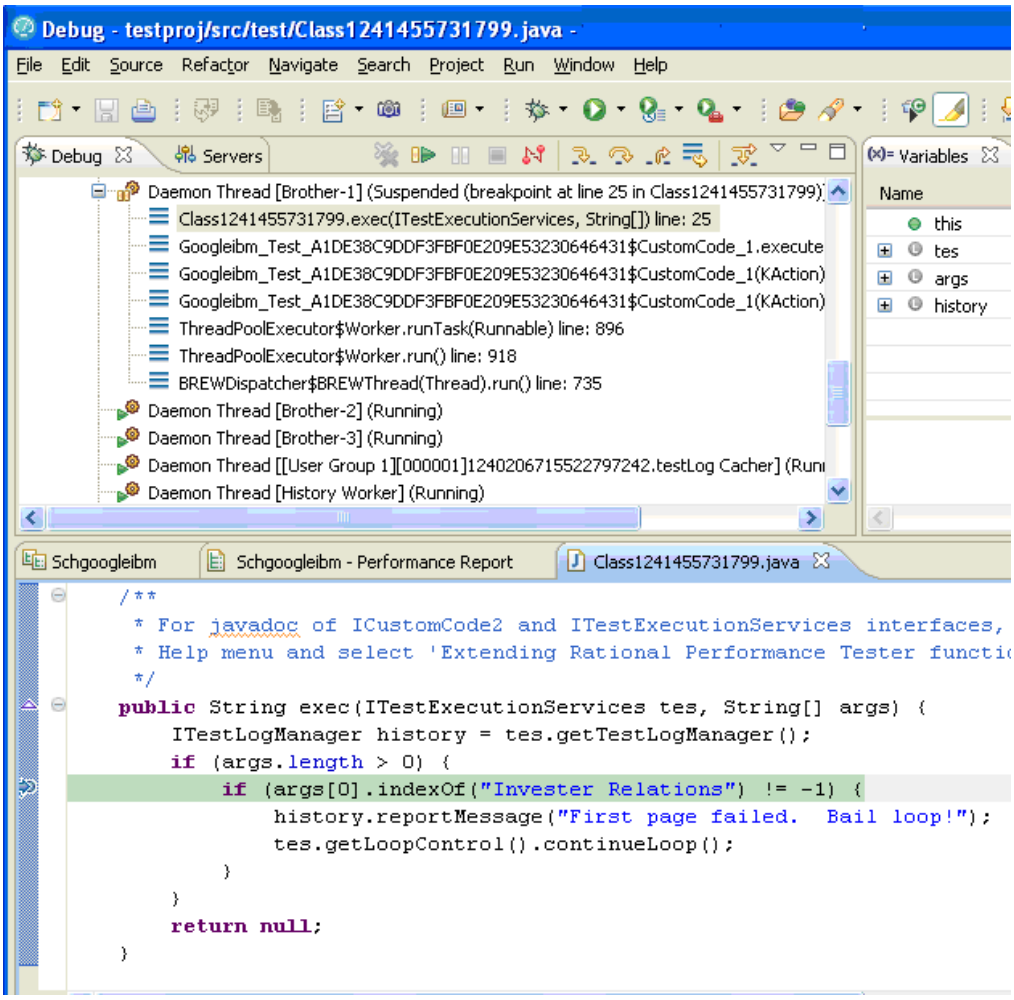
    c. Click **Run > Debug Configurations**.

    d. In the **Debug Configurations** window, right-click **Remote Java Application** and click **New**.

    e. Click **Debug**.
    A list of running threads are displayed in the Debug window and the schedule execution pauses at the debug breakpoint.

    f. If you are doing it for the first time, you might need to provide the source location to see the custom Java code. You do this by taking the following steps:
        i. Click **Edit Source Lookup Path** and click **Add**.
        ii. Click **Workspace Folder > OK**.

iii. Now, expand MyProject, select the src folder, and click **OK**. The schedule run stops at the specified breakpoint.



Reading and writing data from a dataset

When a test is associated with a dataset, you can extend the test either by reading or writing the dataset values from the custom code.

The data that you write into the dataset is saved only when you set **Open mode** to **Shared (for all test executions)** in the **Edit Dataset** window. In other open modes, the modified data is used only for the test run.

The following sample custom code reads and writes the data from the dataset:

```
package datasets;

import java.awt.list;

/**
 * @author IBMHCL Custom Code Samples
 */
```

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

    public myds() {
    }
public String exec(ITestExecutionServices tes, String[] args) {
// the name of the dataset is the same as what is shown in the test. The dataset must be added to the
 test in order
// to get a controller for it.
        IDataSetController control = tes.getDataSetController("/testproj/myds.csv");
        try {
        // once you have the controller you can get a row
        DataSetRow row = control.getNextRow();
        // returns a string representation of the row
        row.getEntireRow();
        // alternatively you can get individual values by the column name
        row.getValue("Column1");

        // you can also write a new row to the dataset
        // -1 means append to the end
        // alternatively you can specify a row number and whether to overwrite that row or to insert a
 new row at the spot
        control.writeRow(-1, Arrays.asList("a", "b", "c"), false);
      } catch (Exception e) {
            tes.getTestLogManager().alwaysReportMessage( e.toString());
            // TODO Auto-generated catch block
            e.printStackTrace();
       }
        return null;
        // or whatever you want to return here
      }
```

Migrating custom code from previous versions

You can run scripts that contain custom code from previous releases and edit tests to make new calls to old or new custom code classes.

About this task

You can perform the following tasks without any additional steps:

- Run a script that contains custom code that was created in a previous release.
- Edit a test to make a new call to an old custom code class.
- Add new custom code to a test that contains old custom code.

To edit a class in existing custom code so that it can call new TestExecutionServices methods, type cast the *IKlog* argument in the old custom code to the *ITestExecutionServices* interface.

When you migrate the custom code from the previous versions, you must use getStatisticsManager2() as getStatisticsManager() API is deprecated from 10.1.0.

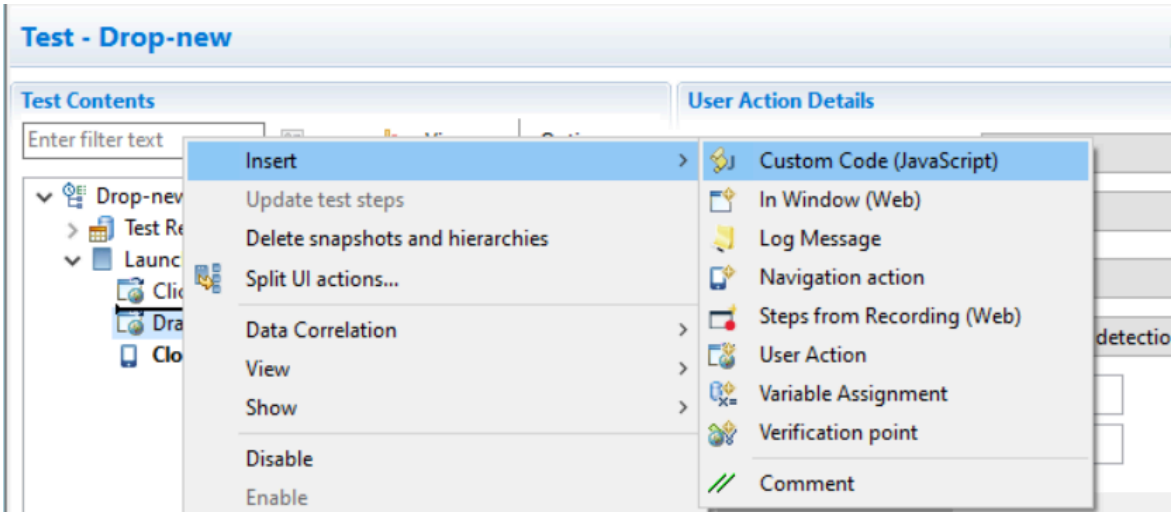
Adding custom JavaScript code as a test step in a Web UI test

You can manually add JavaScript files (*.js) to test scripts with defined functions. You might want to run your own JavaScript snippet such as retrieving some data from the application, doing some actions within the application, or validating some complex logic actions within the application for example. To be able to execute specific code in a test, write your own JavaScript code and insert the custom JavaScript statement as a new test step in your test script.

About this task

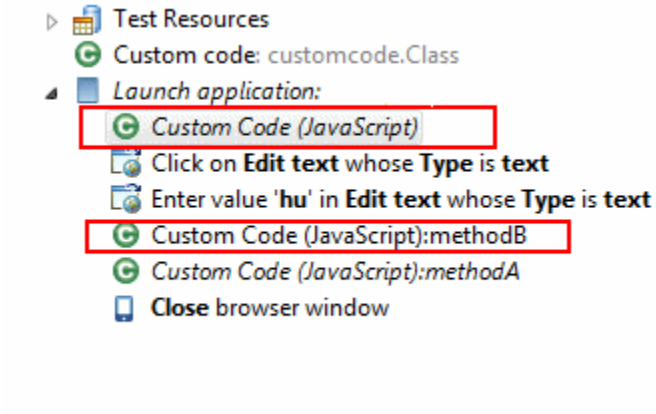
For more information on how to use the JavaScript code in the product, see the Executing JavaScript video .

1. Edit the test script.
2. Select the **Launch application** node and click on **Add** or **Insert** button in the editor and select **Custom Code (JavaScript)**, as shown in the following figure. JavaScript files with .js extension are to be kept in a project of your workspace and must be added a test step with in the launch application node.
   Figure 15. Custom Code (JavaScript) menu



3. In the dialog box that opens, select a JavaScript file to be added to the test step, click OK. It is displayed as links in Referred JavaScripts in the definition pane. A new Test Step is added to the Test script. When a method name is provided, the test step is named **Custom Code (JavaScript):method-name**, otherwise it is named **Custom Code (JavaScript)**, see figure 2.

Figure 16. Custom code added as a test step in a Web UI test



4. Select the step to see the JavaScript custom code definition pane that contains the custom code details. Specify the JavaScript method name to be executed in the **Method** field, and optionally provide the description. Click the **Update** button to add multiple files. The JavaScript custom code will be executed within the Web application. You can also delete the referred JavaScript hyperlink, or click the link to open the JavaScript file in the editor. If the JavaScript method has some parameters to be added, specify the parameters in the **Arguments** field. You can specify the arguments through static text, a variable reference, a dataset reference, or a java custom code.
    a. To enter text values, click **Text** button and enter the text as argument.
    b. To pass test variable reference or dataset reference or JavaScript custom code return value as parameter to JavaScript method, click **Add** button. Select the available data source arguments, datasets, test variables or java custom code. The variable or dataset must be initially created, and a return value added. See example in figure 3.

Figure 17. Custom code details

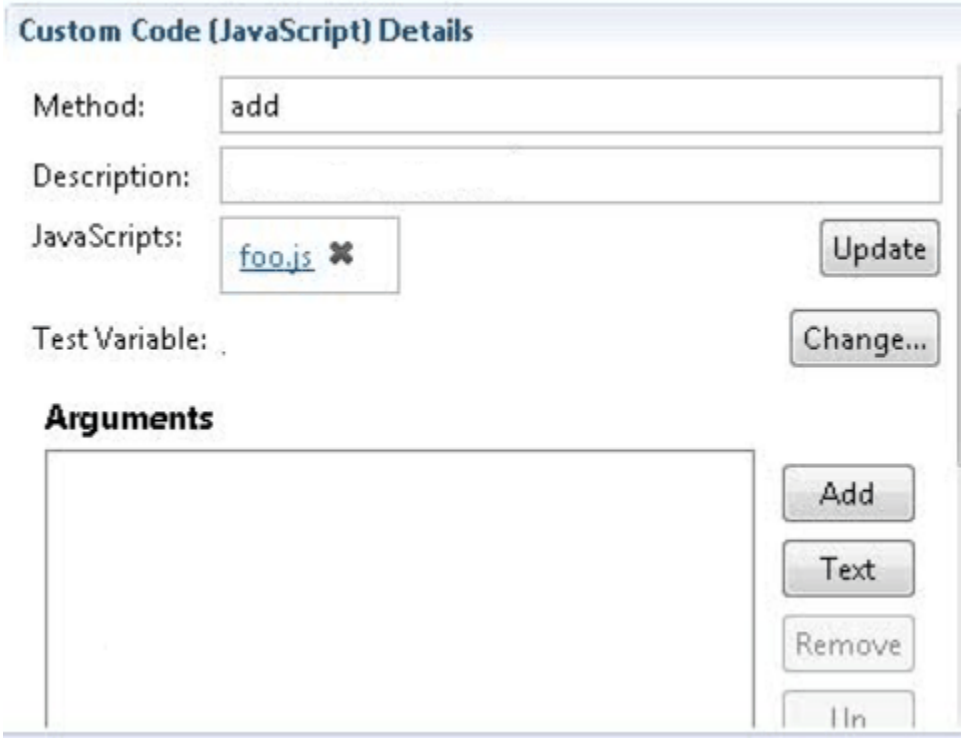
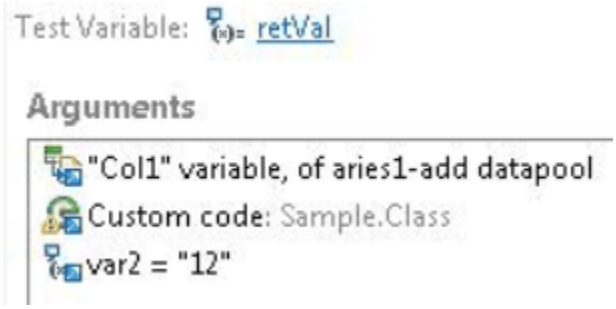


Figure 18. Example of variable and dataset as arguments



5. Run the test script and see the report.
   From version 9.1.1.1, you can replace a JavaScript file with an updated one and apply the changes to all references to the JavaScript file in the test scripts where the file is called, for more details, see Replacing a JavaScript file in a test script on page 632.

Extending the Functional Test perspective

You can use custom code to extend the default functional testing capabilities. You can write custom code and call the code from the test. You can also specify that results from the tests that are affected by your custom code to be included in reports.

You can find the following information:

- Rational Functional TesterHCL OneTest UI proxy SDK on page 1078
- Customizing a script template on page 1166
- Using the API to edit functional test scripts on page 1179

Rational® Functional TesterHCL OneTest™ UI proxy SDK

Extend automated functional testing support for your application's user interface controls.

Introduction to proxy SDK

With Rational® Functional TesterHCL OneTest™ UI proxy software development kit (SDK) you can extend automated functional testing support for your application's user interface controls (GUI test objects), beyond what is provided by Rational® Functional TesterHCL OneTest™ UI by default. The proxy SDK provides detailed documentation, API references and ready-to-use samples and tutorials on how to extend Rational® Functional TesterHCL OneTest™ UI to add support for testing new controls. It also helps you to extend already supported controls from the existing domains. You can develop proxies manually or by using the proxy wizards driven approach.

You can use the Rational® Functional TesterHCL OneTest™ UI proxy SDK to do these tasks:

- Add support for an unsupported control
- Change or enhance what is recorded for an existing control to make playback more resilient
- Add support for sub-items within a control
- Add new data verification tests
- Make an existing control mappable
- Add new properties

Rational® Functional TesterHCL OneTest™ UI proxy SDK supports extending testing support for the following technologies:

- Java
- .NET
- Eclipse
- PowerBuilder
- Visual Basic
- Windows
- MFC
- ActiveX (Support for controls embedded in HTML running on browsers)
- Flex

The current capabilities are as follows:

- Extending current level of support provided for GUI TestObjects
- Adding support for unsupported controls

Note: You can add support for a new control within the scope of currently supported domains and technologies only. Adding support for new domain is currently not supported.

Before you begin

To extend Rational® Functional TesterHCL OneTest™ UI proxies, you need the following:

- Understanding of GUI functional testing
- Detailed understanding of the Rational® Functional TesterHCL OneTest™ UI framework
- Strong familiarity with the application environment
- Knowledge of Java or C# programming based on the proxy framework
- Knowledge of operating system concepts is useful but not necessary

Rational® Functional TesterHCL OneTest™ UI architecture

Rational® Functional TesterHCL OneTest™ UI can be extended to perform additional functions for which it needs to communicate with the application under test (AUT). To do that Rational® Functional TesterHCL OneTest™ UI first establishes a communication channel to the AUT which is called enablement.

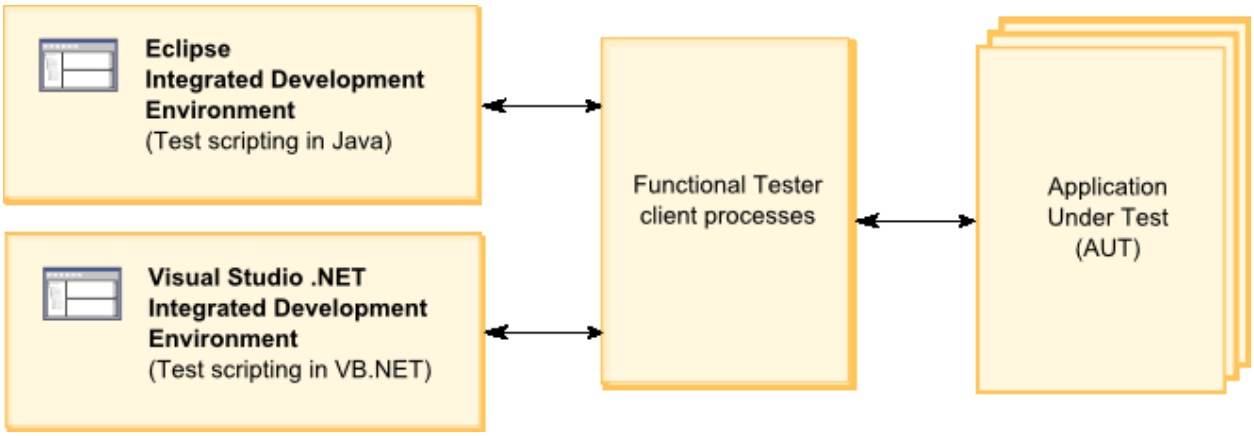
Rational® Functional TesterHCL OneTest™ UI currently supports testing Java™, .Net, HTML, SAP, AJAX, Flex, and native Microsoft® Windows® GUI Controls and each of these supported environments are known as domains. Establishing the communication channel is specific to a domain. You must establish a communication channel for every process and then you need to test the channel. Rational® Functional TesterHCL OneTest™ UI interacts with the AUT process and its controls through the established communication channel to get required information. As part of establishing communication, Rational® Functional TesterHCL OneTest™ UI creates the `DomainImplementation` object instance in the AUT, which in turn abstracts and acts as an interface to provide domain specific details back toRational® Functional TesterHCL OneTest™ UI. The `DomainImplementation` object does the following tasks:

- Gets the top level objects
- Registers available proxy objects that are available for the domain
- Creates the ProxyObject for a given control

Process Model

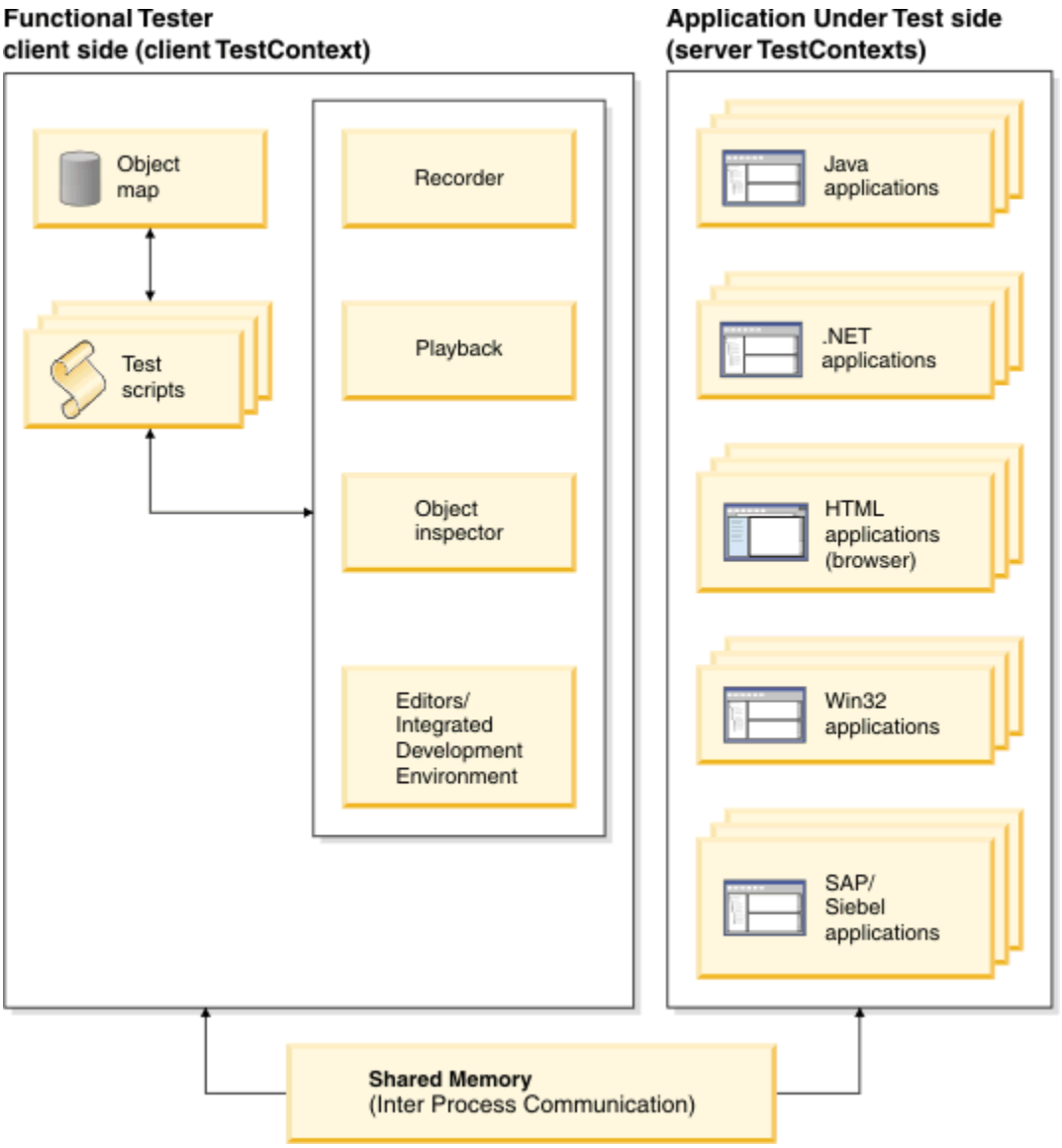
There are two types of Rational® Functional TesterHCL OneTest™ UI processes. The application under Test (AUT) processes are known as Rational® Functional TesterHCL OneTest™ UI server side processes. The recorder, playback, objects inspector, and the IDE (Eclipse or Visual Studio .Net) processes are known as Rational® Functional TesterHCL OneTest™ UI client side processes.

A shared memory inter process communication (IPC) layer complements the high-level components of the Rational® Functional TesterHCL OneTest™ UI process model.



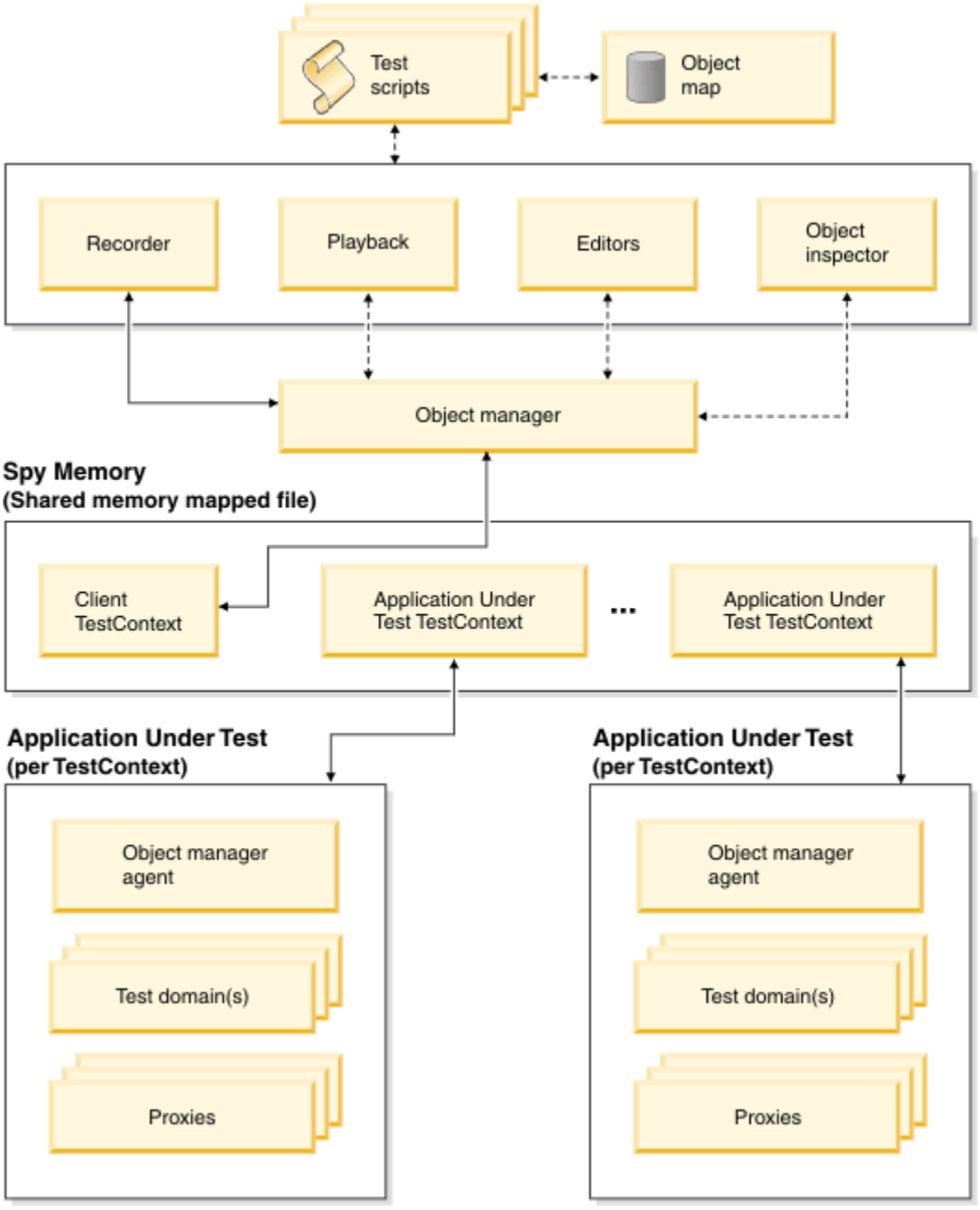
High level interactions

Rational® Functional TesterHCL OneTest™ UI client processes interact with the application under test (AUT) and gather relevant information to perform operations such as recording, playback, and object inspections. This section provides an overview of the interactions between the client processes and the AUT processes. Rational® Functional TesterHCL OneTest™ UI communicates with the AUT through a shared memory inter process communication (IPC) layer.



Rational® Functional TesterHCL OneTest™ UI creates a TestContext object within each process (client or server process) and registers it in the shared memory. The TestContext object is a reference to the process under test. The registered TestContext object is used as a reference for the associated process for any communication. Typically, a TestContext object relates to an operating system level process that can be tested or a test client. It is possible to have more than one TestContext object per process.

Two AUT TestContext objects can not communicate with each other directly. They communicate through the client TestContext object.



Every process in the Rational® Functional TesterHCL OneTest™ UI process model utilizes a TestContext object to manage IPC calls and the requests for that process. Client processes interact with multiple AUT processes, while each AUT process responds to only a single client process at a time. For example, a Find process during playback communicates with all available AUT processes for the TestObject that is being sought.

The ObjectManager handles all server side process communications and meta actions that interact with the AUT. Core record and playback interactions with the AUT originate here and interact with all AUT TestContext objects. An ObjectManager agent handles the ObjectManager meta actions related to a particular TestContext object. Within TestContext objects, TestDomain objects are established to manage Rational® Functional TesterHCL OneTest™ UI

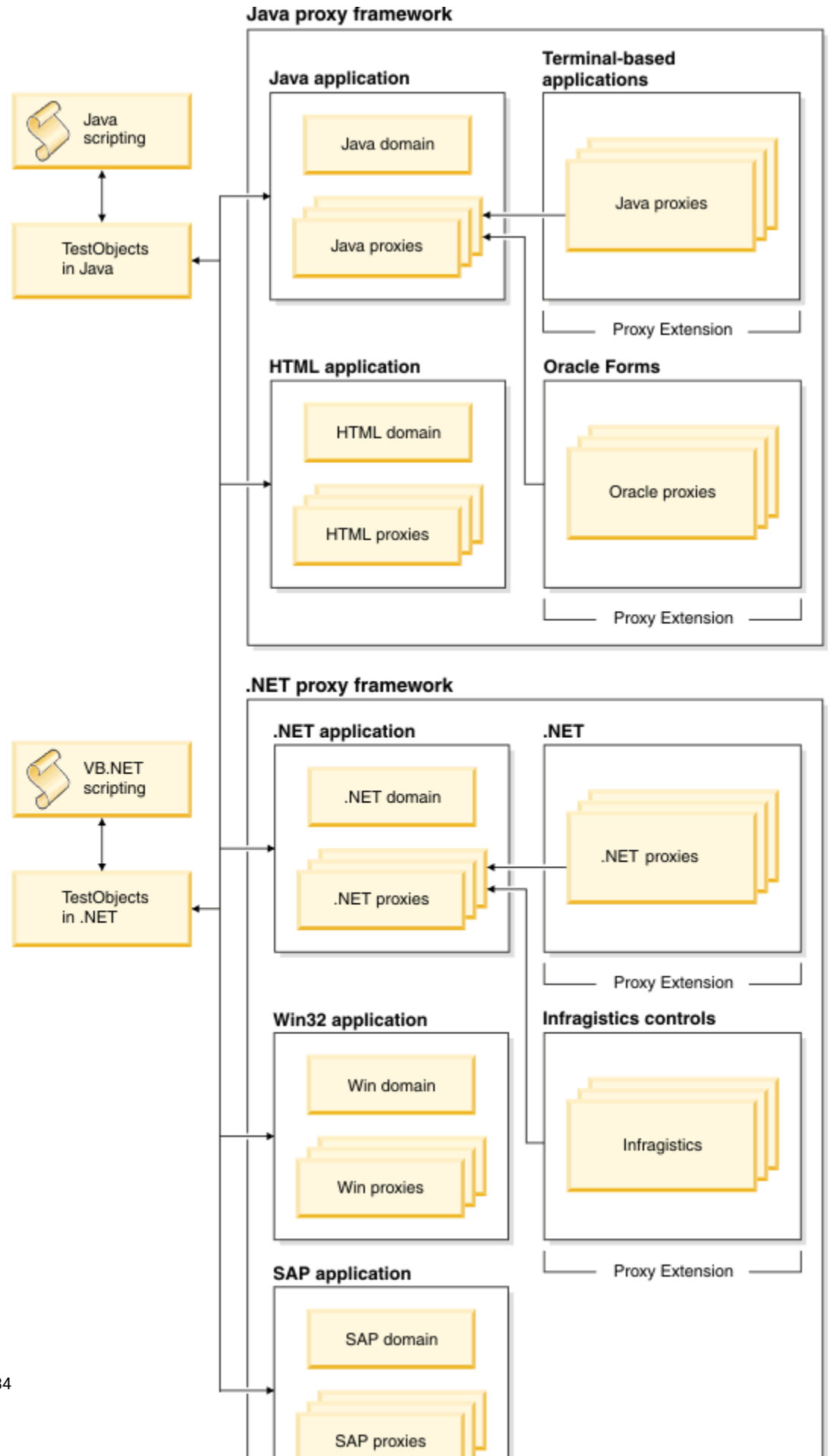
TestDomain-specific communications. For example, for an AUT browser process, a HTML TestContext object and Java™ TestDomain object are established. They are established because browsers contains HTML elements of HTML domain and applets of Java domain.

In a TestDomain object, ProxyTestObjects are created to manage the control level communications. ProxyObjects are created for controls that must be communicated while performing any functional test activity. ProxyObjects have a one-to-one relationship with each control in the AUT. Any interaction between different AUT controls happens through a ProxyTestObject.

Note: TestContext objects, ObjectManager, TestDomain objects, and ProxyObjects are all created in the AUT process.

Proxy model

Rational® Functional TesterHCL OneTest™ UI interacts with application-under-test (AUT) controls through two elements: Proxy objects and Test objects.


Java proxy framework
Java application
Terminal-based applications
Java scripting
Java domain
Java proxies
Java proxies
TestObjects in Java
Proxy Extension
HTML application
Oracle Forms
HTML domain
Oracle proxies
HTML proxies
Proxy Extension
.NET proxy framework
.NET application
.NET
VB.NET scripting
.NET domain
.NET proxies
.NET proxies
TestObjects in .NET
Proxy Extension
Win32 application
Infragistics controls
Win domain
Infragistics
Win proxies
Proxy Extension
SAP application
SAP domain
SAP proxies

Interaction through proxy objects

Proxy objects are similar to wrapper classes for the real controls under test. Any Rational® Functional TesterHCL OneTest™ UI framework communication with AUT controls happen through these proxy objects.

Proxy objects are created and placed where the control under test can be accessed and queried for information. A proxy is written as a Java™ or C# class, which implements the prescribed Rational® Functional TesterHCL OneTest™ UI interface for a GUI test object in the AUT.

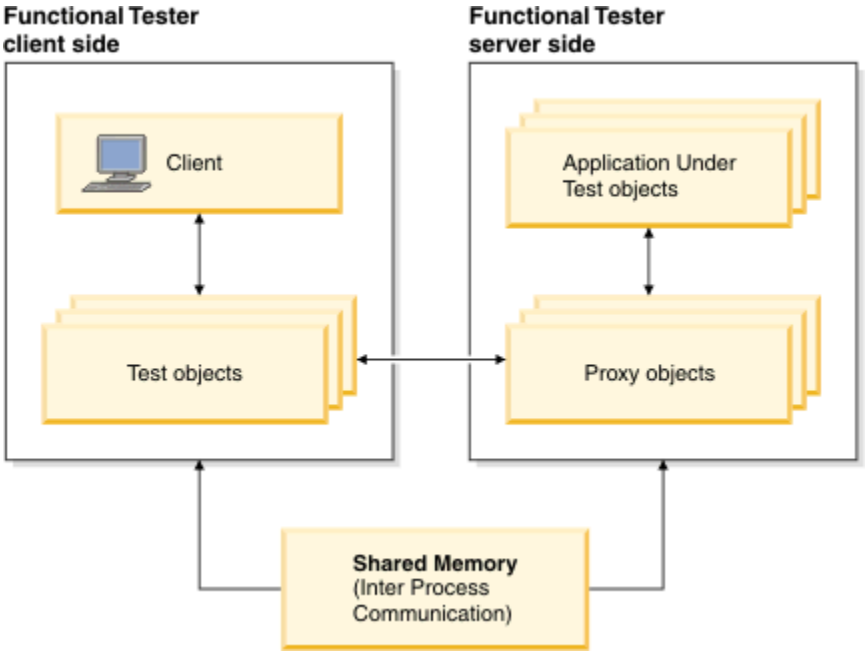
When your application is enabled for testing, the proxy classes are loaded into the application and they become part of it. A proxy object wraps around the actual GUI test object (the native object) in your application, making it testable by Rational® Functional TesterHCL OneTest™ UI.

The Rational® Functional TesterHCL OneTest™ UI framework supports creating a new ProxyObject class or extending any available ProxyObject class to support testing new controls.

Interaction through TestObject

TestObjects are the script-side interface objects for the control under test. A control is exposed as a TestObjects to the test script. For example, a button control is exposed as GuiTestObject. Top level controls like dialogs and frames are exposed as TopLevelTestObject.

The execution of the TestObject methods in turn happens through the corresponding ProxyObject. TestObjects reside in the Rational® Functional TesterHCL OneTest™ UI client side. TestObjects have a reference to the ProxyObject which in turn refers to the AUT control under test.



For each testing environment that Rational® Functional TesterHCL OneTest™ UI supports, such as Java, HTML, .NET, Win32, and SAP, domain objects are established. Under each domain there are ProxyObject classes for all supported AUT controls. The mapping information between ProxyObject classes and AUT controls are stored inside

customization files in the Rational® Functional TesterHCL OneTest™ UI installation directory. These customization files act as a directory for Rational® Functional TesterHCL OneTest™ UI to know which ProxyObject is used for any given AUT control.

Note: The main customization mapping file is `rational_ft.rftcust`. There are other domain specific customization files with the .rftcust extension as well.

ProxyObjects can also be extended to create new ProxyObject classes to support unsupported UI controls. For example, to support the .Net 2.0 DataGridView control, you can create a new proxy class `Rational.Test.Ft.Domain.Net.DataGridViewProxy` and insert the corresponding mapping entry in the `rational_ft.rftcust` file. The following code is an example of the updated section in the customization file.

Exemple

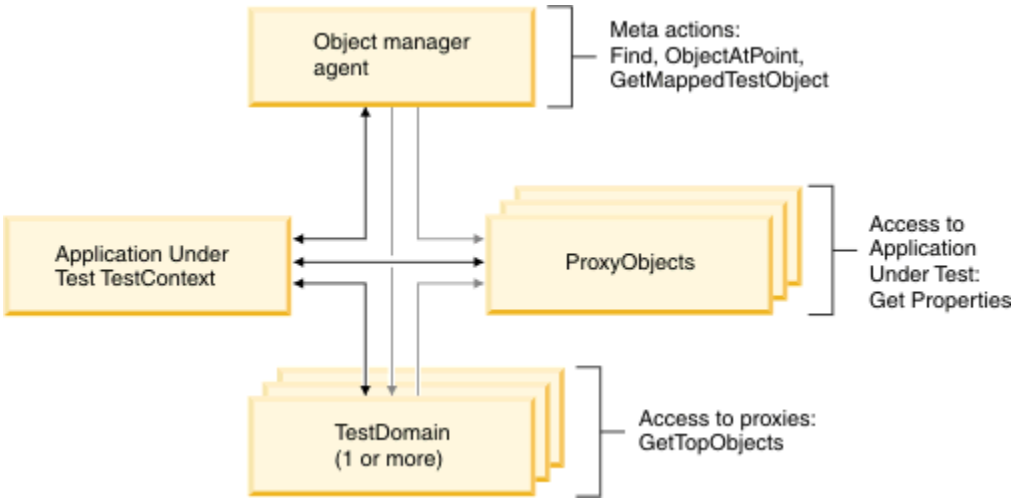
```
<Obj L=".Proxy">
<ClassName>[WhidbeyControls]Rational.Test.Ft.Domain.Net.DataGridViewProxy</ClassName>
 <Replaces/>
  <UsedBy>[System.Windows.Forms]System.Windows.Forms.DataGridView</UsedBy>
</Obj>
```

Application under test interactions

There are several levels of requests in the test process. In each level there are various interactions between Rational® Functional TesterHCL OneTest™ UI and the application under test (AUT).

The following list describes the levels of requests:

- High level requests cause a series of interactions with one or more proxies. Find and ObjectAtPoint are examples of high level requests.
- Direct proxy references can occur for direct access to a proxy. Click and GetTestData are examples of direct proxy references.
- Domain requests which are similar to high level requests, provide access to the hierarchy of base proxies and perform other high level actions.



Recording interactions

The following interactions happen between the Object Manager and Object Manager agents for recording

- Object Manager and Object Manager agents
  - Locate the Object at point. For example, the `proxyAtPoint` method.
  - Get recognition properties and initial references in the map. For example, the `getMappedTestObject` method.
- The following proxy methods are called for recording:

Table 29. Proxy methods for recording

Action	Method
Process low level events	`processMouseEvent`
Locate target of a drag action	`getMethodSpecForPoint`

Verification Point support	`getTestDataTypes`
	`getTestData`
	`getProperties`
	`getStandardProperties`
	`getProperty`

Hierarchy methods	`getMappableParent`
	`getParent`
	`getChildren`

Action	Method
	`getMappableChildren`
	`getOwner`
	`getOwned`
Recognition support	`getRecognitionProperties`
	`shouldBeMapped`
	`getRole`
	`getTestObjectClassName`
	`getRecognitionPropertyWeight`



Playback interactions

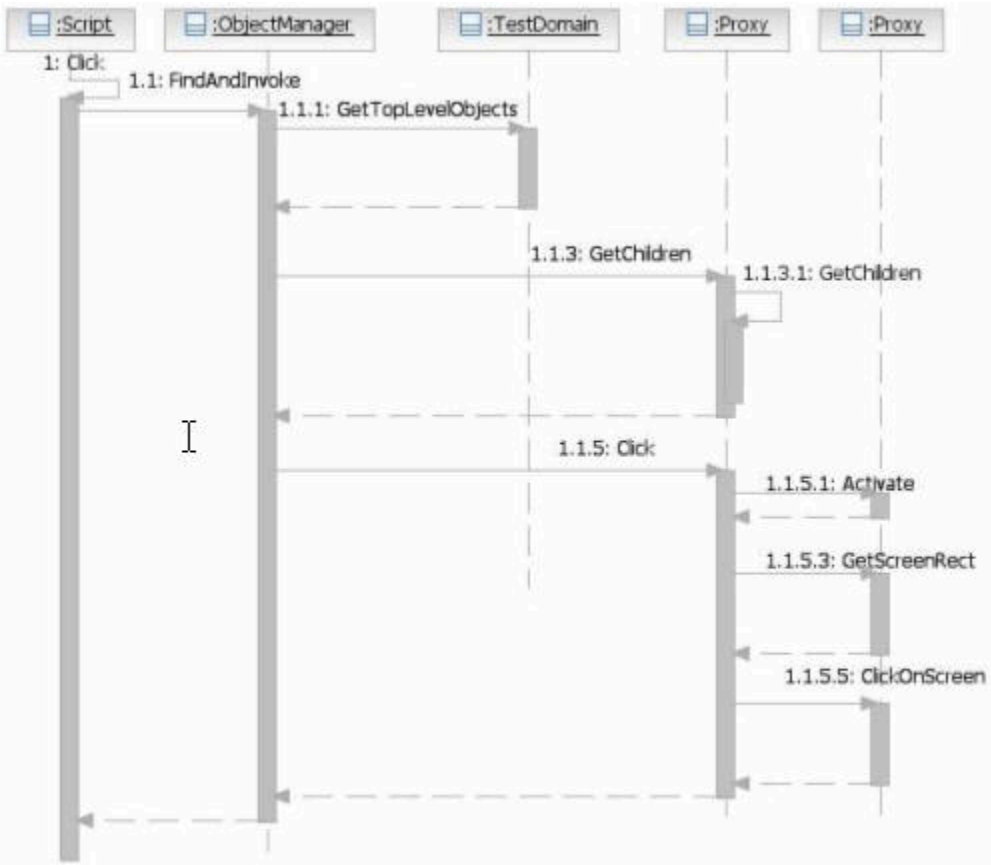
The Rational® Functional TesterHCL OneTest™ UI client sends requests to all test domains if they can find the target object in its recorded object map hierarchy by using the recorded recognition properties.

Table 30. Playback interactions

Result	Action
No target object is found	A TestObject not found exception is thrown.
Several target objects are found	The object finding score is used to determine the winner, or it can be ambiguous.

Table 30. Playback interactions (continued)

Result	Action
A unique TestObject is found	The playback action method is invoked on the proxy. For example, the playback action which was recorded, may be the `click()` method.
More than one TestObject was found within the ambiguity threshold	An ambiguous exception is thrown.



Rational® Functional TesterHCL OneTest™ UI uses recognition properties and control hierarchy to identify a control and provides an interface. This information is collected and stored in the Object Map. During playback the stored information is used to uniquely identify the UI element. Rational® Functional TesterHCL OneTest™ UI also collects information on screen coordinates, control properties and data, reflection details, and portions of the controls when required. It presents the UI element with the gathered information as a TestObject to the script side.

At the time of recording user actions such as mouse clicks, double-clicks or drags are recorded as respective TestObject methods into a test script. For example, `button().click(atPoint(10,10))`. During playback, Rational® Functional TesterHCL OneTest™ UI finds the corresponding TestObject using the information stored in the Object Map and the user action is performed based on it.

Proxy development

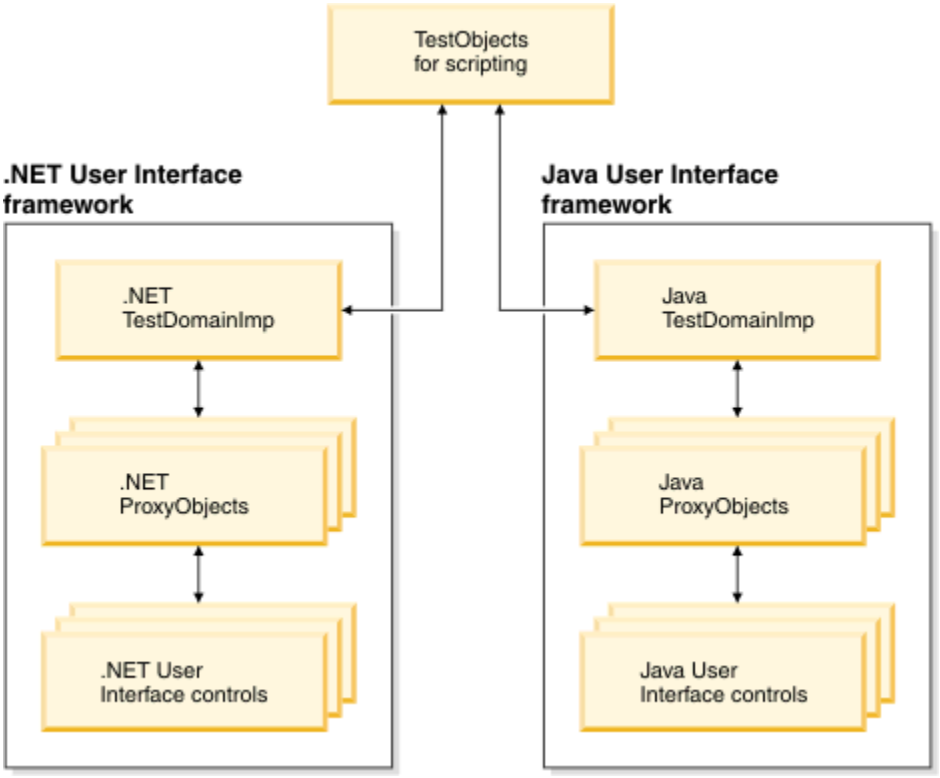
A challenge in functional testing is the variety of user interface (UI) frameworks that are available (for example, Java™ and HTML) and controls (for example, button and table) that testing must support. UI frameworks differ in architecture and programming models and the controls differ in their inheritance hierarchy, methods, properties data, and user actions.

Rational® Functional TesterHCL OneTest™ UI needs to be programmed to add support for different UI frameworks and controls so that appropriate functional testing values can be provided to testers.

The Rational® Functional TesterHCL OneTest™ UI architecture handles the differences in UI frameworks through a respective `TestDomainImplementation` class for different UI frameworks. These `TestDomainImplementation` classes handle the specific properties of each UI frameworks that Rational® Functional TesterHCL OneTest™ UI supports. The following `TestDomainImplementation` classes are available with Rational® Functional TesterHCL OneTest™ UI:

- `JavaTestDomainImplementation`
- `HTMLTestDomainImplementation`
- `NETTestDomainImplementation`
- `WinTestDomainImplementation`
- `SAPTestDomainImplementation`

A `TestDomain` contains a set of controls that is provided by the respective UI framework. Rational® Functional TesterHCL OneTest™ UI understands and handles the differences in controls through `ProxyObject` classes that are implemented for each control or a group of similar controls. A ProxyObject can be seen as wrapper object to the control and is implemented with the standard interfaces that Rational® Functional TesterHCL OneTest™ UI defines. Each proxy method has a specific meaning and Rational® Functional TesterHCL OneTest™ UI calls them at specific times. The ProxyObject returns the details specific to that control. ProxyObject classes handle specifics about each control or a group of similar controls in a supported TestDomain.



Rational® Functional TesterHCL OneTest™ UI offers set of hierarchically grouped ProxyObjects for each supported TestDomains like Java, .Net, Win32, SAP, and HTML. A ProxyObject's inheritance hierarchy in each TestDomain is designed to be the same as the inheritance hierarchy of the control in that TestDomain. Grouping ProxyObjects hierarchically enables you to extend them to create new ProxyObjects when new a control is introduced in the UI framework. You can find the details about available sets of ProxyObjects and controls for each supported UI framework in the ProxyObject hierarchies.

Note: With the current Rational® Functional TesterHCL OneTest™ UI Proxy SDK implementation, you cannot add support for a new UI framework. You can add support for new controls or enhance currently supported controls.

Understanding proxies

A proxy object implements the prescribed Rational® Functional TesterHCL OneTest™ UI interface for a UI control in the application under test (AUT). When you enable your application for testing, the proxy classes are loaded into the application and become part of it. A proxy object wraps around the actual control (the native object) in your application, making it testable in Rational® Functional TesterHCL OneTest™ UI. It is a connection point between the TestObject and the real control (object) being tested at the AUT.

A proxy class is created as a Java™ or C# class depending on the proxy framework that you use and contains information such as how to interact with an object in a particular test domain. The base class for all proxies is ProxyTestObject. ProxyObjects are developed exclusively for a UI control or group of UI controls that require similar

functional testing capabilities. They belong to the same UI framework (test domains) in Rational® Functional TesterHCL OneTest™ UI.

Proxy development environment

Rational® Functional TesterHCL OneTest™ UI proxies can be developed either in Java™ or C# programming languages based on the UI framework of the application under test (AUT).

You must use Java for developing proxies for the following UI frameworks under Java and HTML domains:

- AWT
- Swing
- SWT
- Applet

You must use C# for developing proxies for UI frameworks under the following domains:

- .Net
- Win32

Software requirements

You must have the following software in you computer to develop proxies using the proxy SDK

- Rational® Functional TesterHCL OneTest™ UI 7.0 9.1 or later.
- Eclipse 3.2 or later, or any other JDK for developing proxies in Java
- Microsoft® Visual Studio.Net for developing proxies in C#. For developing ProxyObjects, you must use one of the following integrated development environments:

Note: ProxyObject is exposed to the scripting side of Rational® Functional TesterHCL OneTest™ UI as either .Net or Java TestObject, depending on the test scripting choice.

Setting up proxy projects

ProxyObjects are deployed as either JAR files which are proxies written in Java™, or as Assembly DLLs which are proxies written in C#, along with customization file with the .rftcust extension.

Setting up a proxy project in Eclipse

You can use Eclipse 3.2 or later to develop proxies in Java. To create a new Eclipse Java project:

1. Perform one of the following steps:
   Choose from:
   - From the Eclipse menu, click **File > New > Project**.
   - From the Eclipse toolbar, click **New**, select **Java Project**, and then click **Next**.
2. Type the name of the new project in the **Project name** field and click **Next**.

3. In the Java Settings page:
    a. Click the **Create new source folder** link, and specify **src** as the folder name so that all source files are kept under the `src` directory; then click **Finish**.
    b. Click **Libraries > Add Variable**, and select **RATIONAL_FT_LIB**; then click **OK**.
4. Click **Finish**.

Setting up a proxy project in Visual Studio .Net

You must use Microsoft® Visual Studio .Net to develop proxies in C# for .Net, Win, and SAP domains. To create a new Visual Studio .Net C# project:

1. Create a new project in either of two ways:
   Choose from:
    - From the Visual Studio menu, click **File > New Project**.
    - From the Visual Studio toolbar, click **New Project**.
2. Select **Visual C#** as the project type and **Class Library** as the template to use.
3. Specify the name of the new project and the project location in the **Name** and **Location** fields.
4. Click **OK**.
5. Right-click **References** in the solution explorer and select **Add Reference**.
6. Click **Browse**, and select the `C:\Program Files\IBM\SDP\FunctionalTester\bin\rtxftnet.dll` `C:\Program Files\HCL\HCLOneTest\FunctionalTester\bin\rtxftnet.dll` assembly as a reference to the project.

What to do next

After your project is created and set up, you can start creating your proxy files. You must add a new C# file for each new proxy and similarly for new TestObjects.

Current level of proxy support that Rational® Functional TesterHCL OneTest™ UI provides

To develop proxies for a control, you must understand the current level of support that Rational® Functional TesterHCL OneTest™ UI provides for that control . Consider that you want to add testing support for the Java™ swing UI control, javax.swing.JFormattedTextField.

Verifying that a control already has a specified ProxyObject

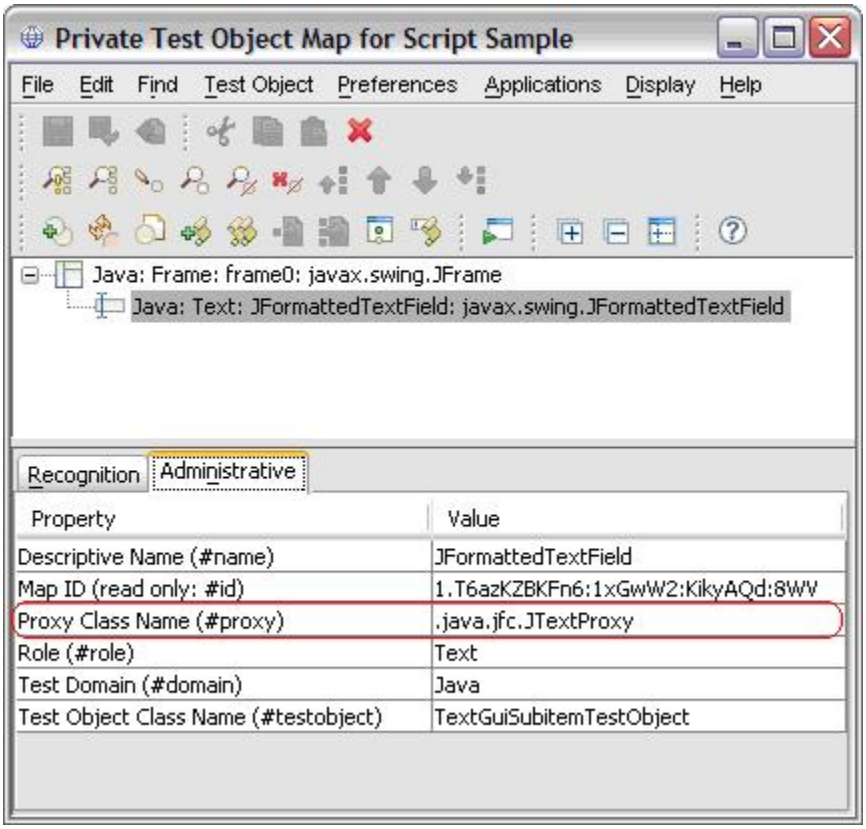
You can identify this by looking at the customization files. All proxy and control mapping information is in the customization files, which have the extension .rftcust. Customization files are either located at the Rational® Functional TesterHCL OneTest™ UI installation directory (`C:\Program Files\IBM\SDP70\FunctionalTester\binC:\Program Files\HCL\HCLOneTest\FunctionalTester\bin`) or the customization directory (`C:\ProgramData\IBM\RFT\customizationC:\ProgramData\HCL\HOTUI\customization`).

For example, if you want to verify whether the control `javax.swing.JFormattedTextField` already has a ProxyObject created for it, search for `javax.swing.JFormattedTextField` in the customization files. If you find a map entry, it means that there is a ProxyObject specifically written for this control.

Verifying the ProxyObject that is currently being used for testing a control

You can verify the ProxyObject by recording a control using Rational® Functional TesterHCL OneTest™ UI. See the Administrative properties of the TestObject in the ObjectMap editor.

For example, open a Java application containing the `JFormattedTextField` control and start recording. Open the ObjectMap editor to view the Administrative properties of the TestObject representing `JFormattedTextField` to find the proxy that Rational® Functional TesterHCL OneTest™ UI is currently using to test this control.



Additional information

To understand the current level of support better, you need some additional information on control hierarchy and customization mapping.

The inheritance hierarchy of an AUT control

The inheritance hierarchy of an application under test (AUT) control is usually found in the UI framework documentation. For example, the inheritance hierarchy of `javax.swing.JFormattedTextField` is available in the Java documentation.

The following list represents the javax.swting.JFormattedTextField inheritance hierarchy

```
java.lang.Object
          java.awt.Component
         java.awt.Container
             javax.swing.JComponent
```

```
                javax.swing.text.JTextComponent
                    javax.swing.JTextField
                        javax.swing.JFormattedTextField
```

Rational® Functional TesterHCL OneTest™ UI customization mapping entry for a proxy

Search for the mapping entry for the currently used ProxyObject in all the customization files. For example, if `java.jfc.JTextProxy` is the currently used ProxyObject for `javax.swing.JFormattedTextField`, the corresponding mapping entry is available at the `rational_ft.rftcust` file as follows:

```
<Obj L=".Proxy">
  <ClassName>com.rational.test.ft.domain.java.jfc.JTextProxy</ClassName>
  <Replaces/>
   <UsedBy>javax.swing.JEditorPane</UsedBy>
   <UsedBy>javax.swing.JTextArea</UsedBy>
   <UsedBy>javax.swing.JTextField</UsedBy>
   <UsedBy>javax.swing.JPasswordField</UsedBy>
   <UsedBy>javax.swing.JTextPane</UsedBy>
</Obj>
```

This example gives you the following information:

- The `java.jfc.JTextProxy` is written exclusively to handle Java swing controls like `JEditorPane`, `JTextArea`, `JTextField`, `JPasswordField`, and `JTextPane`, but not for the `javax.swing.JFormattedTextField` control.
- Rational® Functional TesterHCL OneTest™ UI picked up `java.jfc.JTextProxy` as the ProxyObject for testing `JFormattedTextField` control because `JTextField` is the super class for `JFormattedTextField` and it has `java.jfc.JTextProxy` as the ProxyObject.

  Important: Rational® Functional TesterHCL OneTest™ UI first looks for an explicit ProxyObject and control mapping entry for the control under test. If it fails to find one, it looks for mapping entries for any of the super classes for the control in the inheritance hierarchy.

- Because there is no ProxyObject that is written exclusively for the control `javax.swing.JFormattedTextField`, contol-specific properties such as `formatString` and `unformatedValue` cannot be exposed through the `getProperties()` method of Rational® Functional TesterHCL OneTest™ UI.
- Here is an opportunity for proxy developers to write the `JFormattedTextFieldProxy` by extending `java.jfc.JTextProxy` exclusively for the `javax.swing.JFormattedTextField` control. One of methods that can be extended is `getProperties()` for adding control specific properties like `formatString` and `unformatedValue`.

ProxyObject inheritance hierarchy

You must also look at the proxy inheritance hierarchy as well. Proxy inheritance hierarchy information is available in Rational® Functional TesterHCL OneTest™ UI proxy API reference Guide.

The following list represents the JTextProxy (Functional Tester ProxyObject) inheritance hierarchy.

```
ProxyTestObject
  JavaProxy
    JavaGuiProxy
      awt.ComponentProxy
```

```
        jfc.JComponentProxy
          jfc.JfcGraphicalSubitemProxy
            jfc.JScrollPaneProxy
              jfc.JTextProxy
```

Extending proxies

Rational® Functional TesterHCL OneTest™ UI needs UI control-specific information to perform functional testing operations such as recording, playback, verification points, and data driving. It tries to map the closest proxy if it finds a new control for which it has no proxy.

The Rational® Functional TesterHCL OneTest™ UI architecture enables developers to write a proxy for a particular UI control. Developers can enable Rational® Functional TesterHCL OneTest™ UI to process the specifics of a control by writing proxies. Relevant functional testing capabilities can also be provided.

Proxies provide Rational® Functional TesterHCL OneTest™ UI with details about a control for which it is written. Rational® Functional TesterHCL OneTest™ UI has a predefined set of methods for any proxy and calls each method to get specific details. For any proxy written for a control, these predefined methods are implemented specific to the control.

For any control, Rational® Functional TesterHCL OneTest™ UI provides a set of properties and data types for verification. If the currently provided set of properties and data types is not enough to test the control, a new proxy can be created and by overriding certain methods more properties or data types can be included. The recording behavior of Rational® Functional TesterHCL OneTest™ UI can also be changed by creating new proxies.

Creating a proxy class

You can write a ProxyObject class either in Java™ or C#, depending on which control you are developing the proxy for. Creating a new ProxyObject class is similar to creating a new Java or C# class extending the respective base ProxyObject class.

About this task

To create, build, and deploy a ProxyObject class:

1. Create a ProxyObject Class.
2. Build the ProxyObject binaries using the IDE build commands.

   For Java, the compiled binary is a JAR file.

   Note: The JAR file can also be created through JDK command lines, for example:

   ```
   jar cvf JFormattedTextFieldProxy.jar proxysdk\sample\java\ JFormattedTextFieldProxy.class
   ```

   For .Net, the compiled binary output is a .Net assembly.
3. Map the ProxyObject class to the application under test (AUT) control class.

4. Deploy the ProxyObjects by copying the proxy binary files and customization files to the Rational® Functional TesterHCL OneTest™ UI customization directory (`C:\ProgramData\IBM\RFT\customizationC:\ProgramData\HCL\HOTUI\customization`).

   Rational® Functional TesterHCL OneTest™ UI looks for proxy binary files and customization files in this directory.

5. Restart Rational® Functional TesterHCL OneTest™ UI

Examples: Creating a simple ProxyObject

This example explains how to create a simple ProxyObject.

About this task

To create, build, and deploy the ProxyObject class:

1. Create a ProxyObject class using one of the following methods:

   Choose from:

   - Create a JFormattedTextFieldProxy ProxyObject in Java™ extended from the Rational® Functional TesterHCL OneTest™ UI Java domain proxy, `com.rational.test.ft.domain.java.jfc.JTextProxy`. For example type the following code to create JFormattedTextFieldProxy.java:

     ```
     package proxysdk.samples.java;

     import com.rational.test.ft.domain.java.jfc.JTextProxy;

     /**
      * @author administrator
      *
      * TODO To change the template for this generated type comment go to
      * Window - Preferences - Java - Code Generation - Code and Comments
      */
     public class JFormattedTextFieldProxy extends JTextProxy
     {
      /**
       * Sets the SUT object
       * as a member variable for the proxy.  All interactions with the
       * supplied object are performed through this class.
       */
      public JFormattedTextFieldProxy(Object theObjectInTheSUT)
      {
       super(theObjectInTheSUT) ;
      }

               /*
                *  TODO:  Override more ProxyObject Methods here
                */
     }
     ```

   - Create a `MaskedTextProxy` ProxyObject in C# extended from the Rational® Functional TesterHCL OneTest™ UI .Net domain proxy, `Rational.Test.Ft.Domain.Net.TextBoxProxy`. For example type the following code to create MaskedTextProxy.cs:

```
using Rational.Test.Ft.Domain;
using Rational.Test.Ft.Domain.Net;

namespace ProxySDK.Samples.Net
{
 /// <summary>
 /// Summary description for MaskedTextProxy.
 /// </summary>
 public class MaskedTextProxy:TextBoxProxy
 {
  public MaskedTextProxy(NetTestDomainImplementation domain,
   IChannel channel,
   System.Object theTestObject): base( domain, channel, theTestObject)
  {
  }
  //
  // TODO: Override more ProxyObject Methods here
  //
 }
}
```

- Create a `StatusBarProxy` ProxyObject in C# extended from the Rational® Functional TesterHCL OneTest™ UI Win domain proxy, `Rational.Test.Ft.Domain.Win.GenericProxy`. For example type the following code to create StatusBarProxy.cs:

```
using Rational.Test.Ft.Domain;
using Rational.Test.Ft.Domain.Win;

namespace ProxySDK.Samples.Win
{
 /// <summary>
 /// Summary description for StatusBarProxy.
 /// </summary>
 public class StatusBarProxy:GenericProxy
 {
  public StatusBarProxy(WinTestDomainImplementation domain,
   IChannel channel,
   IWinControl theAUTControl): base( domain, channel, theAUTControl)
  {
   //
   // TODO: Add constructor logic here
   //
  }
 }
}
```

2. Build the ProxyObject binary files using build commands.
   - For Java, the compiled binary output is a JAR file, for example JFormattedTextFieldProxy.jar.
   - For .Net, the compiled binary output is a .Net assembly, for example MaskedTextProxy.dll or StatusBarProxy.dll.
3. Map the ProxyObject classes to the AUT control classes in the Rational® Functional TesterHCL OneTest™ UI customization file (a .rftcust file) using one of the following methods:
   Choose from:

- Specify a mapping entry under the Java domain to use the `proxysdk.samples.java.JFormattedTextFieldProxy` ProxyObject for the `javax.swing.JFormattedTextField` control. For example, type the following code to create JFormattedTextFieldProxy.rftcust:

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfigFile L=".ConfigFile">
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">
   <DomainImplementation L=".DomainImplementation">
    <Name>Java</Name>
    <Obj L=".Proxy">
     <ClassName>proxysdk.samples.java.JFormattedTextFieldProxy</ClassName>
     <Replaces/>
     <UsedBy>javax.swing.JFormattedTextField</UsedBy>
    </Obj>
   </DomainImplementation>
  </Val>
 </Section>
</ConfigFile>
```

  With this mapping, Rational® Functional TesterHCL OneTest™ UI creates a `JFormattedTextFieldProxy` instance for every `JFormattedTextField` Java UI control found.

- Specify a mapping entry under the .Net domain to use the `ProxySDK.Samples.Net.MaskedTextProxy` ProxyObject for the `System.Windows.Forms.MaskedTextBox` .Net control implemented in the `[System.Windows.Forms]` .Net assembly. For example, type the following code to create MaskedTextProxy.rftcust:

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfigFile L=".ConfigFile">
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">
   <DomainImplementation L=".DomainImplementation">
    <Name>NET</Name>
    <Obj L=".Proxy">
     <ClassName>[MaskedTextBoxProxy]ProxySDK.Samples.Net.MaskedTextProxy </ClassName>
     <Replaces/>
     <UsedBy>[System.Windows.Forms]System.Windows.Forms.MaskedTextBox</UsedBy>
    </Obj>
   </DomainImplementation>
  </Val>
 </Section>
</ConfigFile>
```

  Note: Specify assembly names with `[]` while specifying .Net control class names.

- Specify a mapping entry under the Win domain to use the `[StatusBarProxy]ProxySDK.Samples.Win.StatusBarProxy` ProxyObejct for the `StatusBar20WndClass` Win control. For example, type the following code to create StatusBarProxy.rftcust:

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfigFile L=".ConfigFile">
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">
   <DomainImplementation L=".DomainImplementation">
    <Name>Win</Name>
    <Obj L=".Proxy">
     <ClassName>[StatusBarProxy]ProxySDK.Samples.Win.StatusBarProxy</ClassName>
     <Replaces/>
     <UsedBy>STATUSBAR20WNDCLASS</UsedBy>
    </Obj>
   </DomainImplementation>
  </Val>
 </Section>
</ConfigFile>
```

- Specify mapping entries for all ProxyObjects in a combined customization file.

Note: You must specify the proxy mappings in their respective `<DomainImplementation L=".DomainImplementation">` section

For example, type the following code to create combined.rftcust:

```
<ConfigFile L=".ConfigFile">
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">

                          <!--  Add DomainImplementation section for each domain -->

                          <DomainImplementation L=".DomainImplementation">
    <Name>Java</Name>
    <Obj L=".Proxy">
     <ClassName>proxysdk.samples.java.JFormattedTextFieldProxy</ClassName>
     <Replaces/>
     <UsedBy>javax.swing.JFormattedTextField</UsedBy>
    </Obj>
                                     <!--  Add <Obj L=".Proxy"> section here for each Java
 proxy mapping  -->

   </DomainImplementation>

                          <DomainImplementation L=".DomainImplementation">
    <Name>NET</Name>
    <Obj L=".Proxy">
     <ClassName>[MaskedTextBoxProxy]ProxySDK.Samples.Net.MaskedTextProxy </ClassName>
     <Replaces/>
     <UsedBy>[System.Windows.Forms]System.Windows.Forms.MaskedTextBox</UsedBy>
    </Obj>
                                     <!--  Add <Obj L=".Proxy"> section here for each .NET
 proxy mapping  -->

   </DomainImplementation>
```

```
   <DomainImplementation L=".DomainImplementation">
    <Name>Win</Name>
    <Obj L=".Proxy">
     <ClassName>[StatusBarProxy]ProxySDK.Samples.Win.StatusBarProxy</ClassName>
     <Replaces/>
     <UsedBy>STATUSBAR20WNDCLASS</UsedBy>
    </Obj>
                                    <!--  Add <Obj L=".Proxy"> section here for each Win
 proxy mapping  -->

   </DomainImplementation>
  </Val>
 </Section>
</ConfigFile>
```

For any syntax or usage clarification, see the customization file **rational_ft.rftcust** in C:\Program Files \IBM\SDP70\FunctionalTester\bin. C:\Program Files\HCL\HCLOneTest\FunctionalTester\bin. This file contains mapping entries for all ProxyObjects that are delivered with Rational® Functional TesterHCL OneTest™ UI.

4. Deploy the proxy binary files, for example JFormattedTextFieldProxy.jar, MaskedTextProxy.dll, and StatusBarProxy.dll and the corresponding customization files by copying them to the Rational® Functional TesterHCL OneTest™ UI customization directory, C:\ProgramData\IBM\RFT\customization. C:\ProgramData \HCL\HOTUI\customization.
5. Restart Rational® Functional TesterHCL OneTest™ UI.

Adding more control properties

Rational® Functional TesterHCL OneTest™ UI provides a set of control properties for access and property verification. You can add more control properties by extending the `getProperties()` and `getProperty()` APIs.

Before you begin

You can extend the proxy methods that are listed in Table 31: Extensible proxy methods on page 1101:

Table 31. Extensible proxy methods

Java	.Net
java.util.Hashtable getProperties()	System.Collections.Hashtable GetProperties()
Object getProperty(String propertyName)	object GetProperty(string propertyName)

Exemple

The following samples add a new property `ToolTipText`. You can add as many properties as you want in the same manner.

The following sample shows how to add a new property in Java™:

```
import com.rational.test.ft.domain.*;

public class someProxy extends baseProxy
{
```

```
 .
 .
 public java.util.Hashtable getProperties()
 {
    java.util.Hashtable properties = super.getProperties();
    try
    {
 properties.put("toolTipText", getTooltipText());
    }
    catch (Throwable e)
    {
    } // in the odd case we can't get the artifical properties, just ignore them.
    return properties;
 }
 .
 .
 .
 public Object getProperty(String propertyName)
 {
    if (propertyName.equals("toolTipText"))
 return getTooltipText();
    return super.getProperty(propertyName);
 }
}
```

The following sample shows how to add a new property in .Net:

```
using Rational.Test.Ft.Domain;

public class AnyProxy:BaseProxy
{
     .
     .
     .
    public override System.Collections.Hashtable GetProperties()
    {
        System.Collections.Hashtable propertyTable = base.GetProperties();

         if( !propertyTable.Contains("ToolTipText"))
         {
 object text = GetToolTipText();
 if (text != null)
   propertyTable.Add("ToolTipText", text );
         }
         return propertyTable;
    }
    .
    .
    .
   public override object GetProperty(string propertyName)
   {
       object propertyValue = null ;
       if (propertyName == "ToolTipText" )
       {
         propertyValue = GetToolTipText();
       }
       else
       {
```

```
        propertyValue = base.GetProperty(propertyName) ;
      }
      return propertyValue ;
  }
```

What to do next

After successfully developing and deploying this proxy code, a new property `ToolTipText` is added to the control. You can verify this by running the `getProperty("toolTipText")` API on the control.

Adding more data types for a control

Rational® Functional TesterHCL OneTest™ UI provides a set of control data types for data verification point. You can add more control data types by extending the `getTestDataTypes()` and `getTestData()` APIs.

Before you begin

You can extend the proxy methods that are listed in Table 32: Extensible proxy methods on page 1103:

Table 32. Extensible proxy methods

Java	.Net
java.util.Hashtable getTestDataTypes()	System.Collections.Hashtable GetTestData-Types()
ITestData getTestData(String testData-Type)	ITestData GetTestData(string testDataType)

Exemple

The following sample shows how to add a new control data type `Selected Text` in Java™. You can add as many data types as you want in the same manner.

Note: Ensure that the key and value of hash are the same in the hashtable returned by the GetTestDataTypes.

```
public class AnyProxy:BaseProxy
{
     .
     .
     .
    public java.util.Hashtable getTestDataTypes()
    {
         java.util.Hashtable result = super.getTestDataTypes();
         result.put("Selected Text", "Selected Text");
 return result;
    }
    .
    .
    public ITestData getTestData(String testDataType)
    {
 if (testDataType.equals("Text"))
    return createTestDataList(getText());  // getText() method returns text value of the control
 else
```

```
    return super.getTestData(testDataType);
    }
```

The following sample shows how to add a new data type in .Net:

```
Using Rational.Test.Ft.Vp;

public class AnyProxy:BaseProxy
{
  .
  .
  .
  public override System.Collections.Hashtable GetTestDataTypes()
  {
     System.Collections.Hashtable types = base.GetTestDataTypes() ;
     types.Add("Selected Text", "Selected Text") ;
     return types;
  }
  .
  .
  .
  public override ITestData GetTestData(string testDataType)
  {
     ITestData testData = null ;
     switch (testDataType)
     {
        case "Text":
 testData = new TestDataText(((System.Windows.Forms.Control)theTestObject).Text) ;
         break;
      }
      return testData;
  }
}
```

What to do next

After successfully developing and deploying this proxy code, a new control data type `Selected Text` is available while creating a data verification point in the control.

Enhancing the recording behavior

You can enhance the recording behavior for a user action on a control by extending the `processMouseEvent()` API. Rational® Functional TesterHCL OneTest™ UI framework calls the `processMouseEvent()` API when a mouse event happens on a control. The `processMouseEvent()` API tells the Rational® Functional TesterHCL OneTest™ UI framework which method has to be recorded for the mouse action.

Before you begin

For example, when you click a button control, the `button().click()` method is recorded. You can modify this behavior and add more information by extending the `processMouseEvent()` API of the proxy. This API is available only for GUI proxies.

You can extend the methods listed in Table 33: Extensible proxy methods on page 1105:

Table 33. Extensible proxy methods

Java	.Net
void processMouseEvent(IMouseActionInfo action)	void ProcessMouseEvent(IMouseActionInfo action)
void processSingleMouseEvent(IMouseActionInfo action)	void ProcessPreDownMouseEvent(IMouseActionInfo action)
void processHoverMouseEvent(IMouseActionInfo action)	void ProcessPreUpMouseEvent(IMouseActionInfo action)
	void ProcessHoverMouseEvent(IMouseActionInfo action)

Proxy recording APIs

Before you begin

`process`*`xxx`*`MouseEvent()` APIs

There are many process*xxx*MouseEvent() APIs available, but the main API that is used for any mouse event is processMouseEvent(). You can extend the rest of the process*xxx*MouseEvent() APIs the way you want. The following Java™ and .Net implementations of the processMouseEvent() API explain how the rest of the process*xxx*MouseEvent() APIs are related.

The following example shows the Java implementation of processMouseEvent():

```
public void processMouseEvent(IMouseActionInfo action)
{
 int eventState = action.getEventState();
 if ( eventState == IMouseActionInfo.PRE_DOWN ||
   eventState == IMouseActionInfo.PRE_UP ||
   eventState == IMouseActionInfo.POST_UP )
  processSingleMouseEvent(action);
 else if ( eventState == IMouseActionInfo.HOVER )
  processHoverMouseEvent(action);
}
```

The following example shows the .Net implementation of processMouseEvent():

```
public override void ProcessMouseEvent(IMouseActionInfo action)
{
 switch (action.GetEventState())
 {
  case MouseActionStates.PRE_DOWN:
   if (action.GetClickCount() == 1)
    ProcessPreDownMouseEvent(action);
   break;
  case MouseActionStates.PRE_UP:
  // if one click, and it's not a drag, then, we're Done(no need to processPreUpMouseEvent)
   if (action.GetClickCount() != 1 || action.IsDrag())
    ProcessPreUpMouseEvent(action);
    break;
  case MouseActionStates.HOVER:
   ProcessHoverMouseEvent(action);
```

```
   break;
  }
 }
}
```

IMouseActionInfo interface

The processMouseEvent() API obtains the MouseEvent details, such as EventState, screen coordinates, and number of events, and uses these details to decide which method is recorded. The setMethodSpecification() method of IMouseActionInfo is used to return a MethodSpecification() object as a result of the processMouseEvent() API.

MethodSpecification class

The MethodSpecification object represents the method that is being recorded for a particular event. It is initialized with the method name and the parameter is set to the IMouseActionInfo object that is being passed to processMouseEvent(). The recorder picks up this method and records for a given user action.

Exemple

The following sample code extends the processMouseEvent() to change the recording behavior. By default, when you click once, the click() method is recorded. When you double-click, the doubleClick() method is recorded. In this sample the processSingleMouseEvent() API is overridden to swap the recording of click() and doubleClick() methods.

The following sample represents extending the processMouseEvent() in Java:

```
import com.rational.test.ft.domain.IMouseActionInfo;
import com.rational.test.ft.sys.MethodSpecification;
.
.
public void processSingleMouseEvent(IMouseActionInfo action)
{
 String method = null;
 int clicks = action.getClickCount();
 if (clicks == 1)
                 // usually when clicks == 1,  the method is click, now we've changed to method to
 doubleClick
  method = "doubleClick"; //method = "click";
 else if (clicks == 2)
                 // usually when clicks == 2,  the method is doubleClick, now we've changed to method
 to click
  method = "click"; // method = "doubleClick";
 else
  method = "nClick";

 // The method to be recorded is represented using this class in Functional Tester
 MethodSpecification methodSpec = MethodSpecification.proxyMethod(this, method,null);

        // The method for the user action is set here
 action.setActionMethodSpecification(methodSpec);
 }
```

What to do next

After successfully developing and deploying this proxy code, the way that the click() and doubleClick() methods are recorded is swapped.

Enhancing the recording behavior with SubItems

SubItems are Rational® Functional TesterHCL OneTest™ UI defined portions of a control under test. In some cases, you get best results by recording the user interaction with SubItem details rather than recording just with the coordinate information. The disadvantage of using the coordinate information is that when portions of a control are resized or rearranged, playing back the user actions might not return the same results.

Before you begin

For example, in a table control whose column width can be resized, recording the clicks with coordinate is not meaningful during playback if the column width changes.

Rational® Functional TesterHCL OneTest™ UI has a set of predefined SubItems and proxy can use them during recording. During recording, proxy determines the SubItem at a point and sends the SubItem details along with the user action method for the TestObject. At playback time, the proxy again determines the coordinate for the SubItem and the user action is played back.

You can extend the methods that are listed in Table 34: Extensible proxy methods on page 1107:

Table 34. Extensible proxy methods

Java	.Net
System.Collections.ArrayList GetActionArgs(System.Drawing.Point point)	java.util.Vector getActionArgs(java.awt.Point point)
System.Drawing.Rectangle GetSubitemRect(Rational.Test.Ft-.Script.Subitem subitem)	java.awt.Point getScreenPoint(com.rational.test.ft-.script.Subitem subitem)

Recording methods with SubItems

While you are recording an event, the ProcessMouseEvent method is called. Then, the proxy determines the appropriate SubItems at certain points and these SubItems are recorded as part of the event.

Before you begin

The following code is an example of how the event is recorded:

```
listBox.click(atText("Item1"));
```

In this example, click is the event. The `atText("Item1")` parameter is the subItem that the proxy finds at the point. In case of .Net, the GetActionArgs() API returns one or more SubItems of the control. Determining which SubItem to use is specific to the control.

The following example shows the Java™ implementation of recording methods with SubItems:

```
java.util.Vector getActionArgs(java.awt.Point point)
{
 .
 .
       Vector args = new Vector(20);
 SubItem subItem = null;
 IMouseEventInfo event0 = action.getEventInfo(0);
```

```
 Point firstPoint = new Point ( event0.getX(), event0.getY() );
 Point firstPointToList = new Point ( firstPoint.x, firstPoint.y );
 int itemIndex = locationToIndex(firstPointToList);
 String itemText = ((java.awt.List)theTestObject).getItem(itemIndex);
 if (itemText!= null && ! itemText.equals("") )
   subItem = new Text(item);
 else
  subItem = new Index(atIndex);
 .
 .
 args.addElement(subItem);
 .
 .
 }
```

The following example shows the .Net implementation of recording methods with SubItems:

```
protected override System.Collections.ArrayList GetActionArgs(System.Drawing.Point point)
{
 System.Collections.ArrayList args = new System.Collections.ArrayList() ;
 Subitem subitem = null ;
 System.Drawing.Point clientPoint = ((Control)theTestObject).PointToClient(point) ;
 int itemIndex = ((ListBox)theTestObject).IndexFromPoint(clientPoint) ;
        string itemText = ((ListBox)theTestObject).GetItemText(item);

 if (itemText == null || itemText.Length == 0)
 {
         // item has no text so generate an index
  subitem = new Rational.Test.Ft.Script.Index(itemIndex) ;
 }
 if ( subitem != null )
 {
  args.Add(subitem) ;
 }

 return args ;
}
```

Playing back methods with SubItems

During playback, the proxy needs to find the screen coordinate of a SubItem to play back the user action.

Before you begin

The following example shows the Java implementation of playing back methods with SubItems:

```
public void click(MouseModifiers modifiers, Subitem subitem)
{
 .
 .
 Point pt = this.getScreenPoint(subitem);
 new Screen().click( modifiers, pt);
 .
 .
}

public java.awt.Rectangle getScreenPoint (Subitem subitem)
```

```
{
 int index = getItemIndex(subitem);
 if ( index == -1 )
  return null;
       java.awt.Rectangle rectCell = getCellBounds(index);
 java.awt.Rectangle rectScreen = this.getScreenRectangle();
 return new java.awt.Rectangle
 ( rectScreen.x + rectCell.x, rectScreen.y + rectCell.y,
   rectCell.width, rectCell.height );

}
```

The following example shows the .Net implementation of playing back methods with SubItems:

```
protected override System.Drawing.Rectangle GetSubitemRect(Rational.Test.Ft.Script.Subitem subitem)
{
 int index = GetItemIndex(subitem) ;
 return ((ListBox)theTestObject).GetItemRectangle(index) ;
}
```

What to do next

After successfully developing and deploying this proxy code, the recorded methods have the appropriate SubItems and playback happen as expected.

Extending data driving

You must implement the `GetDataDrivableCommand()` method in the proxy to add data driving support to a control. This method returns a method specification to implement data driving support for a control. While using the data driving wizard, the method specification that `GetDataDrivableCommand()` returns is sent to the test script. Proxies can override and return any method that you specify for data driving.

Before you begin

It is not mandatory to add data driving support for every control. Data driving is useful for controls that have common user actions such as a method, and that take data values, such as parameters.

You can extend the methods listed in Table 35: Extensible methods for data driving on page 1109:

Table 35. Extensible methods for data driving

Java	.Net
MethodSpecification getDataDrivableCommand()	MethodSpecification GetDataDrivableCommand()

Exemple

The following sample adds data driving support in Java™:

```
import com.rational.test.ft.domain.*;

public class newProxy extends baseProxy
{
 .
 .
```

```
 public MethodSpecification getDataDrivableCommand()
 {
    if ( !isEditable() )
 return null;
    return MethodSpecification.proxyMethod(
 this, "setText", new Object[]{MethodSpecification.datasetRef(getText())});
  }
 .
 .
}
```

The following sample adds data driving support in .Net:

```
using Rational.Test.Ft.Domain;
using Rational.Test.Ft.Sys;

public class NewProxy:BaseProxy
{
     .
     .
     .
    public override MethodSpecification GetDataDrivableCommand()
    {
       System.String text = GetText();
       if ( text == null )
   text = "";
       return MethodSpecification.ProxyMethod(
  this, "SetText", new System.Object[]{ MethodSpecification.datasetRef(text) } );
    }
    .
    .

}
```

What to do next

After successfully developing and deploying this proxy code, verify it by data driving the control using the Rational® Functional TesterHCL OneTest™ UI data driving wizard. The `TestObject.setText(dpString("text"))` API is inserted into the test script.

Changing the role of a control

The role of a control determines which icon is displayed in the Object Map for a TestObject. You can do this by extending the `getRole()` method and returning one of the Rational® Functional TesterHCL OneTest™ UI predefined role values.

Before you begin

You can extend the methods listed in Table 36: Extensible methods for changing roles on page 1111:

Table 36. Extensible methods for changing roles

Java	.Net
`String getRole()`	`string GetRole()`

The following Java™ sample changes the icon for the TestObject to a slider icon:

Exemple

```
import com.rational.test.ft.domain.*;

public class someProxy extends baseProxy
{
 .
 .
 public String getRole()
 {
     return TestObjectRole.ROLE_SLIDER;
 }

}
```

The following .Net sample changes the icon for the TestObject to a slider icon:

```
using Rational.Test.Ft.Domain;

public class AnyProxy:BaseProxy
{
     .
     .
     public override string GetRole()
     {
           return TestObjectRole.ROLE_SLIDER;
     }
}
```

What to do next

After successfully developing and deploying this proxy code, the icon for the TestObject in the Object map changes to a slider icon.

Modifying the recognition properties and weight of a control

Rational® Functional TesterHCL OneTest™ UI uses recognition properties to uniquely identify a control. Various recognition properties are assigned different weights for recognition analysis. Rational® Functional TesterHCL OneTest™ UI uses these values and weights to identify a control during playback. You can customize the recognition properties and the weights that best suit the controls you are testing.

Before you begin

You can extend the methods listed in Table 37: Extensible methods for recognition properties on page 1112:

Table 37. Extensible methods for recognition properties

Java	.Net
java.util.Hashtable getRecognitionProperties()	System.Collections.Hashtable GetRecognitionProperties()
int getRecognitionPropertyWeight(String propertyName)	int GetRecognitionPropertyWeight(String propertyName)

Note: With the Rational® Functional TesterHCL OneTest™ UI Object Library feature you can also externalize the recognition properties and weights of all controls as an XML file. For more information on Object Library, see the Rational® Functional TesterHCL OneTest™ UI help.

The following Java™ sample adds a new property, `".priorLabel"`, as an additional recognition property.

Exemple

```
import com.rational.test.ft.domain.*;

public class someProxy extends baseProxy
{
 .
 .
 public java.util.Hashtable getRecognitionProperties()
 {
    java.util.Hashtable properties = super.getRecognitionProperties();
    properties.put(".priorLabel", getPriorLabel());
    return properties;
 }
 .
 .
 .
 public Object getRecognitionPropertyWeight(String propertyName)
 {
    if (propertyName.equals(".priorLabel"))
 return 60;
    return super.getRecognitionPropertyWeight(propertyName);
 }
}
```

The following .Net sample adds a new property, `".priorLabel"`, as an additional recognition property.

```
using Rational.Test.Ft.Domain;

public class AnyProxy:BaseProxy
{
     .
     .
     .
    public override System.Collections.Hashtable GetRecognitionProperties()
    {
        System.Collections.Hashtable properties= base.GetRecognitionProperties();
 properties.Add(".priorLabel", GetPriorLabel() );
        return properties;
    }
    .
```

```
    .
    .
   public override object GetRecognitionPropertyWeight(string propertyName)
   {
       if (propertyName == ".priorLabel" )
            return 60;
       return base.GetRecognitionPropertyWeight(propertyName) ;
   }
```

What to do next

After successfully developing and deploying this proxy code, a new recognition property, `".priorLabel"`, is added for the control with 60 as the property weight. You can verify this by looking at the Recognition tab of the TestObject in the Object Map editor.

Changing the mappability of a control

There are certain types of controls in applications under test (AUT) that do not need to be exposed as a TestObject. For example, container controls have no useful testing value and are not exposed. Rational® Functional TesterHCL OneTest™ UI needs these container controls to run certain methods to retrieve information about their children.

Before you begin

You can specify whether to expose a control as a TestObject by extending the `ShouldBeMapped()` method. By default, only GUI TestObjects are mapped.

For example, the Panel control is not mapped. If you want to map this control, however, extend the `ShouldBeMapped()` method and specify the return value as true.

You can extend the methods listed in Table 38: Extensible methods for mapping TestObjects on page 1113:

Table 38. Extensible methods for mapping TestObjects

Java	.Net
boolean shouldBeMapped()	bool ShouldBeMapped()

Exemple

The following Java™ sample uses the `ShouldBeMapped()` method to change the mappability of a control:

```
import com.rational.test.ft.domain.*;

public class someProxy extends baseProxy
{
 .
 .
 public boolean shouldBeMapped()
 {
    return true;
 }
}
```

The following .Net sample uses the `ShouldBeMapped()` method to change the mappability of a control:

```
using Rational.Test.Ft.Domain;

public class SomeProxy:BaseProxy
{
  .
  .
  public override bool ShouldBeMapped()
  {
     return true;
  }
}
```

What to do next

After successfully developing and deploying this proxy code, the control for which the proxy is written for will be mapped.

Mapping proxies to controls

Rational® Functional TesterHCL OneTest™ UI identifies each application under test (AUT) control by its class name. Running the `testObject.getProperty(".class")` method in the test script gives you the class name. The mapping is established through the class names of the respective control and ProxyObject.

Before you begin

To map proxies to controls, you must explicitly map newly developed ProxyObjects to a control or group of controls through an external map file called customization files with the .rftcust extension. Rational® Functional TesterHCL OneTest™ UI refers these customization files and creates ProxyObject instances for a control as specified in the mapping. You must create your own customization file to specify the mapping information between the ProxyObject and the control. You can deploy the newly created customization file by saving it in the Rational® Functional TesterHCL OneTest™ UI customization directory.

About this task

To map a proxy class to an AUT control class, add the `ClassName` and `UsedBy` tags within the `DomainImplementation` start and end tags.

Note: You must add your proxy class name within the `ClassName` tag and the name of the AUT class that the proxy represents within the `UsedBy` tag.

Exemple

This customization file, for example, shows a mapping entry:

```
<DomainImplementation L=".DomainImplementation">
<Name>Java</Name>
<ClassName>com.rational.test.ft.domain.java.awt.JSpinnerProxy</ClassName>
<Replaces/>
<UsedBy>java.awt.JSpinner</UsedBy>
</DomainImplementation>
```

What to do next

While updating the customization file, make sure that the file meets the following conditions:

- Make no typing mistakes while specifying the class. The strings are case sensitive.
- Use fully qualified class names. Use the complete class name including the package separated with a period (.).
- Match the domain type of the proxy to the section in the customization file by name.
- Use an appropriate XML format:
    - Nest start and end tags properly.
    - Specify names for tags (case is significant).

Customization file

You can specify extensible components such as proxies, TestObjects, values, and value managers in an external customization file with the .rftcust extension. After the files are deployed, these extended components become part of the Rational® Functional TesterHCL OneTest™ UI framework. The main customization file, `rational_ft.rftcust` is located in the Rational® Functional TesterHCL OneTest™ UI installation directory or the customization directory.

You can create many customization files. When you start Rational® Functional TesterHCL OneTest™ UI, it reads all the customization files and stores the details in the shared memory. Further references to the customization files are made in the shared memory

Customization file syntax

The customization file is an XML file with many sections that are marked with <Section></Section> tags. Each section has a name and contains the content between the tags. The following example shows a basic section:

```
<Section L=".ConfigFileSection">
<Name>Section Name</Name>
.
.
Section content
.
</Section >
```

Sections are optional and you can also insert empty sections also in a customization file. Each section has its own syntax. For section components that require implementation in both component models (Java™ and .NET) there should be two <ComponentModel> tags, one for Java and .NET each:

```
<ComponentModel L=".ComponentModel">
<Name>Java</Name>
</ComponentModel>
<ComponentModel L=".ComponentModel">
<Name>Net</Name>
</ComponentModel>
```

The `proxies` section is the most commonly used section. It contains a `<DomainImplementation>` tag for each test domain for specifying proxy classes that are deployed and the associated application under test (AUT) class names for which the proxy is used. Within the `<DomainImplementation>` tag, the `<Obj L=".Proxy">` tags are used for each proxy class that is established and the `<UsedBy>` tags specify the class name of the AUT control.

You can have more than one `<UsedBy>` tag for a single proxy class if you want to use the same proxy for similar controls.

You must use the `<Section>` tags appropriately, meeting their requirements and extension components to be deployed into the Rational® Functional TesterHCL OneTest™ UI framework.

Complete syntax of the main customization

The code shown below is the complete syntax for the main customization file.

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

 <--Proxy Section: all the proxies are defined here domain wise-->
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">
   <DomainImplementation L=".DomainImplementation">
    <Name>Java</Name>
    <Obj L=".Proxy">
     <ClassName></ClassName>
     <Replaces/>
     <UsedBy></UsedBy>
     .
     .
    </Obj>
    .
    .
   </DomainImplementation>
   <DomainImplementation L=".DomainImplementation">
    <Name>Net</Name>
    <Obj L=".Proxy">
     <ClassName></ClassName>
     <Replaces/>
     <UsedBy></UsedBy>
     .
     .
    </Obj>
    .
    .
   </DomainImplementation>
   <DomainImplementation L=".DomainImplementation">
    <Name>Win</Name>
    <Obj L=".Proxy">
     <ClassName></ClassName>
     <Replaces/>
     <UsedBy></UsedBy>
     .
     .
    </Obj>
```

```
   .
  .
  </DomainImplementation>
  .
  .
 </Val>
</Section>


<--ValueManager Section: all newly defined Value and Valuemanager classes are defined here -->
<Section L=".ConfigFileSection">
 <Name>valueManagers</Name>
 <Val L=".ValueManagerManager">
  <ComponentModel L=".ComponentModel">
   <Name>Java</Name>
   <Obj L=".ValueManager">
    <Id></Id>
    <ValueClass></ValueClass>
    <Manager></Manager>
   </Obj>
  </ComponentModel>
  <ComponentModel L=".ComponentModel">
   <Name>Net</Name>
   <Obj L=".ValueManager">
    <Id></Id>
    <ValueClass></ValueClass>
    <Manager></Manager>
   </Obj>
  </ComponentModel>
 </Val>
</Section>

<--Value Converter Section: -->
<Section L=".ConfigFileSection">
 <Name>valueConverters</Name>
 <Val L=".ValueConverterManager">
 </Val>
</Section>

<--Property Converter Section: -->
<Section L=".ConfigFileSection">
 <Name>propertyConverters</Name>
 <Val L=".PropertyConverterManager">
  <ComponentModel L=".ComponentModel">
   <Name>Java</Name>
   <Obj L=".PropertyConverter">
    <Property></Property>
    <Converter></Converter>
   </Obj>
  </ComponentModel>
  <ComponentModel L=".ComponentModel">
   <Name>Net</Name>
   <Obj L=".PropertyConverter">
    <Property></Property>
    <Converter></Converter>
   </Obj>
  </ComponentModel>
 </Val>
```

```
 </Section>

 <--Options Converter Section: -->
 <Section L=".ConfigFileSection">
  <Name>options</Name>
  <Val L=".Options">
   <Obj L=".Option">
    <Name></Name>
    <Type></Type>
    <ReadOnly></ReadOnly>
    <DefaultValue/>
    <Description></Description>
    <Label/>
    <Category></Category>
    <LegalValues/>
   </Obj>
  </Val>
 </Section>

 <-- RoleMap Section: -->
 <Section L=".ConfigFileSection">
  <Name>roleMap</Name>
  <Val L=".RoleMap">
   <Role L=".Role">
    <Name></Name>
    <Icon></Icon>
   </Role>
  </Val>
 </Section>

 <--TestObject Section: canonical name for all the newly created testobjects defined here -->
 <Section L=".ConfigFileSection">
  <Name>testObjects</Name>
  <Val L=".TestObjectManager">
   <ComponentModel L=".ComponentModel">
    <Name>Java</Name>
    <Obj L=".TestObject">
     <CanonicalName></CanonicalName>
     <TestObject></TestObject>
    </Obj>
    .
    .
   </ComponentModel>
   <ComponentModel L=".ComponentModel">
    <Name>Net</Name>
    <Obj L=".TestObject">
     <CanonicalName></CanonicalName>
     <TestObject></TestObject>
    </Obj>
    .
    .
   </ComponentModel>
  </Val>
 </Section>

</ConfigFile>
```

Deploying a proxy

After you have developed the proxy binary and customization files, you must deploy them for the changes to take effect.

About this task

To deploy your customized proxies, copy the compiled binary files and customization files to the product customization directory.

Note: You can find the location of the customization directory in the system registry key `HKEY_LOCAL_MACHINE\Software\Rational Software\Rational Test\8\Rational FT Customization Directory` `HKEY_LOCAL_MACHINE\Software\HCL Technologies\HCL OneTest UI\HCL FT Customization Directory`. The default location for the customization directory in Windows 7 is `C:\ProgramData\IBM\RFT\customization.` `C:\ProgramData\HCL\HOTUI\customization.`

Debugging the proxy code

Debugging the proxy code is an essential part of the proxy development process for problem determination. The proxy code containing the JAR file or .Net assembly file with the .dll extension is loaded into the application under test (AUT) process. To debug the proxy code, attach the respective debugger to the AUT process after the proxy binary files are loaded into the AUT.

Preparing the debug environment

About this task

Before you debug the proxy code, perform the following tasks:

- Save the debug version of the proxy binary files in the customization directory and restart Rational® Functional TesterHCL OneTest™ UI.
- In case of Java™, enable the JRE that the AUT uses with the Rational® Functional TesterHCL OneTest™ UI enabler.
- In case of .Net, start the .Net AUT and record a click action on any control using Rational® Functional TesterHCL OneTest™ UI to enable it for testing.

Note: The proxy assembly .dll file is loaded only when the first click is recorded on Windows® and .Net applications. Windows and .Net applications are enabled dynamically for testing. In case of Java, the Application Configurator Tool enables the AUT.

Setting invocation timeout

Debugging proxies is time sensitive and the invocation times out after two minutes by default. To adjust timeout for debugging, add a DWORD value InvocationTimeout in milliseconds under HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\RationalTest\8\Options HKEY_LOCAL_MACHINE\SOFTWARE\HCL Technologies\HCL OneTest UI in the Windows registry. A timeout during debugging throws a SpyMemory MutexTimeout exception.

Debugging record

The getChildAtPoint() method is the entry point for proxy debugging for record. Any user action calls the processMouseEvent() method, even before AUT sees the event. Rational® Functional TesterHCL OneTest™ UI processes the user actions, for example whether the action is a click or drag and accordingly the method specification and arguments are generated. For best results, use these methods to start inserting breakpoints.

Debugging playback

The getMappableChildren() method is the entry point for proxy debugging for playback. During proxy development, most ObjectNotFound problems that occur result from a mismatch between the object hierarchy that the record produces and the hierarchy produced during playback. Make sure that the getMappableParent() and getMappableChildren() methods are symmetrical.

Implementing logs for proxy code debug

Rational® Functional TesterHCL OneTest™ UI provides a log infrastructure that you can use while debugging the developed proxy code. The FTDebug class is available in both Java™ and .Net proxy development frameworks. You can instantiate an object of the FTDebug class for each proxy class and log any information, warning, or error message categorically.

Before you begin

This example code shows how to implement logging for the proxy code in Java:

```
import com.rational.test.ft.util.FtDebug;
.
public class MyProxy extends BaseProxy
{
 protected static FtDebug debug = new FtDebug("myproxies");
 .
 void anyMethod()
 {
  debug.trace("Beginging of anyMethod()");
  .
  debug.verbose("I'm doing this!");
  .
  debug.warning("Not critical, good to have it fixed");
  .
  debug.error("I shouldn't have been here!!") ;
  .
  debug.trace("End of anyMethod()");
 }
}
```

This example code shows how to implement logging for the proxy code in .Net:

```
.
using Rational.Test.Ft.Util;
.
public class MyProxy : BaseProxy
{
 protected static FtDebug debug = new FtDebug("myproxies");
```

```
 .
 void anyMethod()
 {
  debug.Trace("Beginging of anyMethod()");
  .
  debug.Verbose("I'm doing this!");
  .
  debug.Warning("Not critical, good to have it fixed");
  .
  debug.Error("I shouldn't have been here!!") ;
  .
  debug.Trace("End of anyMethod()");
 }
}
```

In this example, a new instance of the FtDebug() class, called `myproxies`, is created. You can control the level of trace information emitted during execution, using the Trace Components settings on the Logging and Tracing page.

Extending proxies for Flex custom controls

Main features of specific support for Flex custom control

You can extend support for testing Adobe® Flex application by using proxy SDK wizards.

Main features of specific support for Flex custom control are:

- Control is mapped to a more meaningful proxy and test object.
- Roles can be assigned to the control.
- Recognition properties can be added.
- Data verification point and data driven test can be implemented.
- Recording is control specific. For example, you now record on the same control:
  `list_FlexCustomControl_FlexCustomControl1().select("Food");`
- If the `getMethodSpec` method is not overridden to emit the methods as required then the events in the superclass is generated as `performAction()`.
- `getTestObjectClassName` method must be overridden to point to the correct test object.

Exemple

Example

For more information about specific support for Flex custom control, you must refer to the **Flex custom control samples** topics in the information centre.

Flex custom control support for Proxy SDK wizard

Rational® Functional TesterHCL OneTest™ UI supports testing functional aspects of Adobe® Flex custom controls in a generic and specific way. Proxies can be created and deployed using the proxy SDK wizard. Certain changes must be done regarding the base class for proxy and the test object. These are given in the sample proxy and test object. Once the deployment is done and the proxy is mapped to the control, you can test the specific control. You can extend

Rational® Functional TesterHCL OneTest™ UI capabilities by using the Proxy SDK wizard to test the Flex custom control.

Before you begin

- Copy the contents of ClassInfo tag in the FlexCustom.xml file available in the sample directory into the FlexEnv.xml file in the bin directory of the product install.
- Create a file without extension in `C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust`. Add the path of the application directory into this file.

About this task

If you want to map a specific control to a proxy and have the data verification point and the dataset support, the specific custom control support helps you achieve it. The basic requirements are:

1. You must write a delegate for the custom control and map the events and properties in the FlexEnv.xml file. Delegate is an actionscript class which allows automation framework to understand the events from the control. References are available in Flex Builder directory where a delegate exists corresponding to each standard control. For more information see, Flex Data Visualization Developer's Guide in the Adobe site.
2. Creating a proxy project and a test object. Associate the proxy and the test object with the control for which the proxy is written. The attached project has .jar containing the FlexCustomControlProxy and FlexCustomControlObject.
3. You must map the proxy to the control in the *.rftcust file generated using the proxy SDK wizard.

Main features of specific support for Flex custom control:

About this task

- Control is mapped to a more meaningful proxy and test object. For example, FlexCustomControl control gets mapped to FlexCustomControlProxy and FlexCustomControltestObject
- Role can be assigned to the control. For example, the given custom control has role “List? assigned to it.
- Recognition properties can be added.
- Data verification point and data driven test can be implemented.
- Recording is control specific. For example, you now record on the same control:
  `list__FlexCustomControl_FlexCustomControl1().select("Food");`
- `getMethodSpec` method is overridden to emit the methods as required otherwise it goes to superclass and event might be generated as `performAction()`. See FlexCustomControlProxy.java
- `getTestObjectClassName` method must be overridden to point to the correct test object.

Developing proxies using the Proxy SDK wizard

Rational® Functional TesterHCL OneTest™ UI proxy SDK supports wizard driven development of proxies. Proxy development life cycle stages such as creating a proxy project, creating a proxy class, exporting the proxy packages and deploying these proxies is done using the proxy wizards.

To use the proxy development wizard, you must have:

- Detailed understanding of the Rational® Functional TesterHCL OneTest™ UI framework.
- Knowledge of the controls for which proxies are created.
- Knowledge of Java programming based on the proxy framework.

Rational® Functional TesterHCL OneTest™ UI Proxy SDK wizard supports developing proxies for Flex, Eclipse and Java technologies

Creating a proxy project

The Rational® Functional TesterHCL OneTest™ UI proxy SDK supports wizard driven proxy development. This wizard helps you create a new proxy project in the work area. Proxy classes that are developed will be stored in this proxy project container. When a proxy project is created, an entry class for that proxy is created. For example, proxyproject.rftcust is created in the proxy project folder.

Before you begin

To use the proxy development wizard, you must switch to the Java perspective.

About this task

To create a proxy project:

1. Click **File > New > Other**.
2. In the **Select a wizard** dialog box, expand **Functional Test > Proxy Development**, and double-click **Proxy Project**; then click **Next**.
3. Type the project name in the **Project name** field.
4. You can browse to the folder and save the project in an appropriate location.

Creating a proxy class

Proxy classes that you develop are stored in the proxy project container. You must use proxy-class wizard development to create a proxy framework. The proxy writer must provide the logic for the proxy. For example, you must enter the logic for the methods and interfaces for the proxy to recognize the controls.

Before you begin

To use the proxy-development wizard to create a proxy class, you must switch to the Java perspective.

About this task

To create a proxy class:

1. Click **File > New > Other**.
2. In the **Select a wizard** dialog box, expand **Functional Test > Proxy Development**, and double-click **Proxy Class**; then click **Next**.

   Result

   The **FT Proxy Creation** window opens.

3. In the **Source folder** field, type the source folder name for the new class. You can also click **Browse** to select a new source folder.
4. In the **Package** field, type a valid package name. You can also click **Browse** to select a package. Package names are updated and can be accessed subsequently while creating a new proxy class.
5. **Required:** In the **Proxy Class Name** field, type a valid proxy class name.
6. **Required:** In the **Control Class Name** field, type a valid class name for the control for which you are developing the proxy. The class name must be a valid name with the package information in it.
7. In the **Domain Name** list, Java is selected as the default domain. Use the default selection.
8. **Optional:** Select a method stub that you want to create, and select the access modifiers for the new class.
9. Type a valid class name in the **Superclass** field. You can also click **Browse** to select a superclass for this class. You can use superclasses to extend proxies. You can extend proxies by using the existing proxy classes.
10. **Optional:** Click **Add** to choose interfaces that the new class implements; then click **Finish**.
    Result

    Note: If you want to select a feature method for a proxy, click **Next**. The **Proxy Feature Page** window opens.

11. **Optional:** In the **Available Methods** section, select the feature to add to the proxy. For example, click **Recording**, and expand the recording tree. Feature methods that pertain to the recording features are displayed. Select the method to add.
    Result

    Tip: To select all the available methods, click **Select All**.

12. Click **Finish**.
    Result
    A proxy class is created in the proxy project.

Exporting proxy packages

You can use the export wizard to export resources to an archive file. The components that must be deployed are bundled in a file.The deployable assets are exported to the target location. You can select the required proxy package items to export and save the items in the target location. These assets can be used later in different computers. For example, the .rftcfgjar file can be deployed to computers that run the Microsoft Windows operating systems.

Before you begin

To use the proxy development wizard, you must switch to the Java perspective.

About this task

To export the resources to an archive file:

1. Click **File > Export**.
   Result
   The Export wizard opens.
2. Click **Functional Test > Functional Test configuration/customization to a JAR file**, and then click **Next**.
3. Select the proxy package to export. In the **File** text field, type the file name. You can also click **Browse** to select the destination path.
4. Click **Finish**.
   Result
   The .rftcfgjar file is exported to the specified location.

Importing proxy packages

The import wizard imports the proxy package to the customization directory of Rational® Functional TesterHCL OneTest™ UI. The import wizard displays the proxy items that are available in the proxy package. You can import these items into Rational® Functional TesterHCL OneTest™ UI or any computer.

Before you begin
To use the proxy development wizard, you must switch to the Java perspective. You must also have an archived proxy package from which to import.

About this task
To import items from an archive file:

1. Click **File > Import**.
   Result
   The Import wizard opens.
2. Click **Functional Test > Functional Test configuration and customization items**, and then click **Next**.
   Result
   The **Import configuration items** window opens.
3. In the **Import from** field, browse for the archive file in the file system.
   Result
   The **Select the items to be imported** window opens.
4. The items that the archive file contains are displayed in the **Select the items to be imported** window. Select the items to import, and click **Finish**.
   Result
   The selected items are displayed in Rational® Functional TesterHCL OneTest™ UI.

Proxy project creation wizard

This wizard helps you create a new proxy project in the work area. When you first open the New Project wizard, you need to select the type of project you want to create. To assist in locating a particular wizard, the text field can be used to show only the wizards that match the entered text.

The following fields must be typed to create a proxy project using the proxy wizard:

- **Project name**
- **Location**
- **Finish**

Project name

Type the name of the new proxy project that you want to create.

Location

Type the location of the new proxy project that you want to create.

Finish

Save the changes and close the proxy project creation wizard.

Proxy class creation wizard

The New Proxy Class wizard helps you to create a new proxy class in an existing proxy project. This wizard creates a proxy stub. The proxy writer provides the logic for the proxy. For example, you must specify the logic for the methods and interfaces to modify the existing behavior of Rational® Functional TesterHCL OneTest™ UI.

Table 39. Proxy Creation window

Option	Description	Default
Source folder	Specify a source folder for the new proxy class. Either type a valid source folder path or click **Browse** to select a source folder via a dialog.	The source folder of the element that was selected when the wizard started.
Package	Specify a package to contain the new class. Either type a valid package name or click **Browse** to select a package.	The package of the element that was selected when the wizard started.
Proxy Class Name	Type a valid name for the new proxy class.	No default name is provided.
Control Class Name	Type a valid control class name for the proxy being developed, for example, `java.awt.Button`.	No default name is provided.
Modifiers	Select one or more access modifiers for the new class. Choose **public**, **default**, **private**, **protected**, **abstract**, **final**, **static**. Note that **private**, **protected**, and **Static** are available only if you specify an enclosing type.	`public`
Superclass	Type the name of a superclass or click **Browse** to select a superclass for this class.	The type (not the compilation unit) that was selected when the wizard started or `java.lang.Object`

Table 39. Proxy Creation window (continued)

Option	Description	Default
Interfaces	Click **Add** to choose interfaces that the new class implements.	No interfaces are selected by default.
Which method stubs would you like to create?	Choose the method stubs to create in this class: • **Public static void main(String [] args)** adds a main method stub to the new class. • **Constructors from superclass** copies the constructors from the superclass of the new class and adds these stubs to the new class. • **Inherited abstract methods** adds to the new class stubs of any abstract methods from superclasses or methods of interfaces that need to be implemented.	Inherited abstract methods enabled.

Table 40. Proxy Feature Page window

Option	Description	Default
Available Methods	Methods pertaining to particular features such as Recording , Playback are displayed in this section.	blank

Exporting proxy items

This wizard helps you to archive and export proxy items to a destination folder.

The export wizard presents three tasks:

- **Select items to export**
- **File**
- **Finish**

Select items to export

Select proxy project items to archive.

File

The .rftcfgjar file is exported to the file location specified in this field.

Finish

Save the changes and close the export wizard.

Importing proxy items

This wizard displays the proxy items in the archive file and helps you select the proxy items and deploy those items in an existing project.

The **Import configuration items** window contains the **Import from** field. Specify the file from which you want to import proxy items. Type the full path or browse to select the path on the file system.

The **Select the items to be imported** window contains these elements:

Select items to import

Select the proxy items from the proxy package to import in your project.

Finish

Save the changes and close the import wizard.

TestObjects

TestObjects are the script-side interfaces for proxies and application under test (AUT) controls. A TestObject is a connection point between the test script and a ProxyObject that connects to the real object in the AUT. During recording, statements are recorded and objects are added to the Object Map. The script uses the information from the Object Map to construct and find TestObjects.

For example, if you record `Button().click()`, the `Button()` method finds an object that is based on the mapped properties and binds the TestObject to an object in the AUT. This binding is required to query information from the actual object, such as asking the button directly where it is on the screen. Then, the `click()` method is executed and the TestObject is unregistered, which releases the connection to the actual object in the AUT. Using TestObjects from the map in this manner manages the lifetime of the object automatically.

TestObjects are exposed to the scripting side based on which proxy it is mapped to. You must specify the proxy and TestObject mapping so that when a control is exposed to the script, the control is exposed as the TestObject that is specified in the mapping. You can create the mapping between a proxy and TestObject by overriding `getTestObjectClassName()` method on any proxy. If you want to change the TestObject that is mapped to a proxy, override the `getTestObjectClassName()` API to return the canonical name that is specified in the customization file.

Role of a TestObject

TestObjects are wrapper classes to proxies for TestScripts. A control is exposed as a Java™ or C# object to TestScripts through TestObjects. TestObjects are implemented in both Java and C#, because Rational® Functional TesterHCL OneTest™ UI supports using both Java and Visual Basic .Net as the test-script language. If you use Visual Basic .Net scripts, TestObjects that are implemented in C# are used. For Java scripts, TestObjects that are implemented using Java are used.

TestObjects forward method calls to the respective proxy using the `InvokeProxy` method, as shown in the following example. The method is actually implemented in the proxy.

Exemple

```
public virtual void PerformClick() {
InvokeProxy("performClick");
}

public void Click(Rational.Test.Ft.Script.Index subitem) {
InvokeProxyWithGuiDelay("click", "(L.script.Index;)", new System.Object[]{subitem});
}
```

In this example, the `"PerformClick()"` method of TestObject calls the `performClick()` method of the proxy. The proxy carries out the actual playback operation of `performClick()`.

Adding a new TestObject

You can add a new TestObject when there are no TestObjects with the method that you want to expose for a control, available within the set of predefined TestObjects provided by Rational® Functional TesterHCL OneTest™ UI. For example, you can expose any button control as `GuiTestObject()` and operations such as `click()` and `doubleClick()` are defined as methods in it. You can create a new TestObject if you want to introduce a new method call such as `myClick()`, which is not defined in any of the existing TestObjects.

Before you begin



Note: Proxies are developed either in Java™ or in C#. However, you must implement TestObjects for proxies in both Java and C#, because they are just wrappers to proxies in both Java and Visual Basic .Net scripts. When you add a new custom TestObject, ensure that you define the Java implementation of the TestObject. This is required even if you use VS.NET IDE for creating functional test scripts.

About this task

To add a new TestObject:

1. Create the constructors for the TestObject.

   Note: Every TestObject must have five standard constructors. New methods that are defined follow these constructors.

2. Define new canonical names for the TestObject in the customization file.

   You must specify two entries for every canonical name for both Java and .NET TestObjects in the customization file, because they are developed in both Java and .NET.

3. Map the proxies to the newly created TestObject.
4. Build the TestObject binary files.

   You can group all the Java TestObject binary files in a single JAR file and the .NET TestObjects in a single .NET assembly.

5. Deploy the TestObject binary files by copying the files to the Rational® Functional TesterHCL OneTest™ UI customization directory, `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization`.
6. Restart Rational® Functional TesterHCL OneTest™ UI.

What to do next

After successfully developing and deploying the TestObject binary files, the Administrative properties of the newly recorded controls, for which you created new TestObjects show the new TestObject names under Test Object Class Name.

Examples: Adding a new TestObject

This example explains how to add a new TestObject.

About this task

To create, build, and deploy the TestObject:

1. Create the constructors for the TestObject.

   This example code shows the Java™ TestObject:

```
package sdk.sample;

import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.TestObjectReference;
import com.rational.test.ft.object.map.SpyMappedTestObject;

public class ExtendedGuiTestObject extends GuiTestObject{

 // FIVE Standard Contructors,  has to be defined in every new TestObject

 public ExtendedGuiTestObject(SpyMappedTestObject mappedObject)
 {
  super(mappedObject);
 }

 public ExtendedGuiTestObject(SpyMappedTestObject mappedObject, TestObject anchor)
 {
  super(mappedObject, anchor);
 }

 public ExtendedGuiTestObject(SpyMappedTestObject mappedObject,
         TestObject anchor,
         long scriptCommandFlags)
 {
  super(mappedObject, anchor, scriptCommandFlags);
 }

 public ExtendedGuiTestObject(TestObjectReference ref)
 {
  super(ref);
 }

 public ExtendedGuiTestObject(TestObject obj)
```

```
 {
  super(obj);
 }


       // Newly added Method for this TestObejct,  just a call forwarder using invokeProxy API

 public void performClick()
 {
  invokeProxy("performClick");
 }
}
```

This example code shows the .Net TestObject:

```
using TestObjectReference = Rational.Test.Ft.Object.TestObjectReference;
using Rational.Test.Ft.Object.Interfaces;
using Rational.Test.Ft.Object.Manager;
using Rational.Test.Ft.Object.Map;

namespace SDK.Sample
{

 public class ExtendedGuiTestObject:GuiTestObject
 {
  // FIVE Standard Contructors,  has to be defined in every new TestObject
  public ExtendedGuiTestObject(SpyMappedTestObject mappedObject):base (mappedObject) {
  }

  public ExtendedGuiTestObject(SpyMappedTestObject mappedObject, TestObject anchor)
   :base (mappedObject, anchor){
  }

  public ExtendedGuiTestObject(SpyMappedTestObject mappedObject, TestObject anchor,
   long scriptCommandFlags):base (mappedObject, anchor, scriptCommandFlags) {
  }

  public ExtendedGuiTestObject(TestObjectReference ref_Renamed):base (ref_Renamed) {
  }

  public ExtendedGuiTestObject(TestObject obj):base (obj) {
  }

  // Newly added Method for this TestObejct,  just a call forwarder using InvokeProxy API

  public virtual void PerformClick() {
   InvokeProxy("performClick");
  }
 }
}
```

2. Define new canonical names for the TestObject in the customization file.

   This example shows how you can define new canonical names for a TestObject:

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

```
 <Section L=".ConfigFileSection">
  <Name>proxies</Name>
  <Val L=".ProxyManager">
   <DomainImplementation L=".DomainImplementation">
    <Name>Net</Name>
    <Obj L=".Proxy">
     <ClassName>[NETProxyExtension]SDK.Sample.TestButtonProxy</ClassName>
     <Replaces/>
     <UsedBy>Rational.Controls.CustomButton</UsedBy>
    </Obj>
   </DomainImplementation>
  </Val>
 </Section>
 <Section L=".ConfigFileSection">
  <Name>testObjects</Name>
  <Val L=".TestObjectManager">
   <ComponentModel L=".ComponentModel">
    <Name>Java</Name>
   </ComponentModel>
   <ComponentModel L=".ComponentModel">
    <Name>Net</Name>
    <Obj L=".TestObject">
     <CanonicalName>ExtendedGuiTestObject</CanonicalName>
     <TestObject>[NETProxyExtension]SDK.Sample.ExtendedGuiTestObject</TestObject>
    </Obj>
   </ComponentModel>
                         <ComponentModel L=".ComponentModel">
    <Name>Java</Name>
    <Obj L=".TestObject">
     <CanonicalName>ExtendedGuiTestObject</CanonicalName>
     <TestObject>sdk.sample.ExtendedGuiTestObject</TestObject>
    </Obj>
   </ComponentModel>
  </Val>
 </Section>
</ConfigFile>
```

3. Map proxies to the newly created TestObject.

   This example shows the Java proxy source overriding `getTestObjectClassName()` method:

```
import com.rational.test.ft.domain.*;
.
.
public String getTestObjectClassName()
{
    return "ExtendedGuiTestObject";  //  the canonical name for the newly created testObject
}
```

   This example shows the .Net proxy overriding `GetTestObjectClassName()` method:

```
using Rational.Test.Ft.Domain;.
.
public override System.String GetTestObjectClassName()
{
   return "ExtendedGuiTestObject";  //  the canonical name for the newly created testObject
}
```

4. Build the TestObject binary files.
5. Deploy the TestObject binary files by copying the files to the Rational® Functional TesterHCL OneTest™ UI customization directory, `C:\ProgramData\IBM\RFT\customizationC:\ProgramData\HCL\HOTUI\customization`.
6. Restart Rational® Functional TesterHCL OneTest™ UI.

Mapping proxies to TestObjects

You can extend a proxy so that Rational® Functional TesterHCL OneTest™ UI uses a different a TestObject to provide a suitable interface on the scripting side.

Before you begin

Note: Rational® Functional TesterHCL OneTest™ UI contains a set of TestObjects with predefined methods that you can reuse while creating new proxies. Rational® Functional TesterHCL OneTest™ UI defined TestObjects have canonical names that are associated with them. For the complete list of predefined TestObjects and their associated canonical names, see the `com.rational.test.ft.domain.ProxyTestObject` or `Rational.Test.Ft.Domain.ProxyTestObject` members. Canonical names are string names for fully qualified TestObject class names.

You can extend the proxy methods that are listed in Table 41: Extensible methods for mapping proxies to TestObjects on page 1133:

Table 41. Extensible methods for mapping proxies to TestObjects

Java	.Net
String getTestObjectClass-Name()	String GetTestObjectClass-Name()

The `GetTestObjectClassName()` returns the canonical name of the TestObject that needs to be mapped for a proxy. The TestObject can be either a new or existing TestObject.

The following Java™ sample returns the canonical name of the TestObject as `TOGGLEGUITESTOBJECT_CLASSNAME`:

Exemple

```
import com.rational.test.ft.domain.*;

public class someProxy extends baseProxy
{
 .
 .
 public String  getTestObjectClassName()
 {
    return ProxyTestObject.TOGGLEGUITESTOBJECT_CLASSNAME;
 }
 .
```

```
 .
}
```

The following .Net sample returns the canonical name of the TestObject as `TOGGLEGUITESTOBJECT_CLASSNAME`:

```
using Rational.Test.Ft.Domain;

public class AnyProxy:BaseProxy
{
     .
     .
     .
    public override String GetTestObjectClassName()
    {
        return ProxyTestObject.TOGGLEGUITESTOBJECT_CLASSNAME;
    }
    .
    .

}
```

What to do next

After successfully developing and deploying this proxy code, the control that is being recorded is mapped to the new TestObject. You can verify this by looking at the TestObjectName property under Administrative properties of the TestObject.

ProxyObject hierarchy

While developing new proxies by extending existing proxies for any domain, it is important to understand the existing proxies and the hierarchies, so that you extend the right proxy.

For more information on methods and properties of proxy classes, see Proxy API reference.

Identifying the control under test

The first task for a functional testing tool is to identify the control under test and represent it in the application under test (AUT). Rational® Functional TesterHCL OneTest™ UI identifies that a control is using recognition properties and the hierarchy, and represents it in the TestObject Map. Recognition properties and hierarchy might vary for different controls. The proxy developed for a control provides this information to Rational® Functional TesterHCL OneTest™ UI. While developing new proxies you are going to inherit the base proxies in each domain. Usually you will not override these methods, because they are already implemented in the base proxies.

Recognition properties

You can use the methods that are listed in Table 42: Extensible methods for specifying recognition property on page 1135 to specify recognition properties and weight for a control.

Note: You can also specify recognition properties and weight using the Object Library.

Table 42. Extensible methods for specifying recognition property

Java	.Net
Hashtable getRecognitionProperties()	Hashtable GetRecognitionProperties()
int getRecognitionPropertyWeight(String propertyName)	int GetRecognitionPropertyWeight(String propertyName)

Hierarchy

You can use the methods that are listed in Table 43: Extensible methods for specifying hierarchy on page 1135 to specify the hierarchy of a control.

Note: Typically, these methods are implemented in the base proxy classes and you may not need to extend.

Table 43. Extensible methods for specifying hierarchy

Java	.Net
getParent()	GetParent()
getTopParent()	GetTopParent()
getChildren()	GetChildren()
Object getChildAtPoint(Point pt)	Object GetChildAtPoint(Point pt)
getOwner()	GetOwner()
getOwnedObjects()	GetOwnedObjects()

Defining

You can use the methods that are listed in Table 44: Extensible methods for changing administrative characteristics on page 1135 to change the administrative characteristics of a control, for example icons or descriptive names.

Table 44. Extensible methods for changing administrative characteristics

Java	.Net
String getTestObjectClassName()	String GetTestObjectClassName()
String getDescriptiveName()	String GetDescriptiveName()
String getUniqueId()	String GetUniqueId()

Table 44. Extensible methods for changing administrative characteristics (continued)

Java	.Net
boolean shouldBeMapped()	bool ShouldBeMapped()
String getRole()	String GetRole()
String getObjectClassName()	String GetObjectClassName()

Recording

Rational® Functional TesterHCL OneTest™ UI records the user actions performed on a control when the recorder is on. These user actions can be grouped as mouse interactions and keyboard interactions. You can use the methods that are listed in Table 45: Extensible methods for recording on page 1136 for recording.

Table 45. Extensible methods for recording

Java	.Net
void processMouseEvent(IMouseActionInfo action)	void ProcessMouseEvent(IMouseActionInfo action)
void processPreDownMouseEvent(IMouseActionInfo action)	void ProcessPreDownMouseEvent(IMouseActionInfo action)
void processPreUpMouseEvent(IMouseActionInfo action)	void ProcessPreUpMouseEvent(IMouseActionInfo action)
void processPostUpMouseEvent(IMouseActionInfo action)	void ProcessPostUpMouseEvent(IMouseActionInfo action)
void processHoverMouseEvent(IMouseActionInfo action)	void ProcessHoverMouseEvent(IMouseActionInfo action)
getScriptCommandFlags()	GetScriptCommandFlags()
Vector getSubItems()	ArrayList GetActionArgs(Point pt)
Rectangle getRectangle(SubItem)	SubItem FindSubItem(Point pt)

Playback

You can use the methods that are listed in Table 46: Extensible methods for playback on page 1137 to find the screen rectangle or point for a SubItems.

Note: Typically, you do not need to extend these methods unless you introduce new SubItems.

Table 46. Extensible methods for playback

Java	.Net
Rectangle getScreenRectangle(SubItem)	Point GetPointForSubItem(SubItem)

Verification points

You can add more datatypes and properties to a control.

Data verification

You can use the methods that are listed in Table 47: Extensible methods for adding datatypes on page 1137 to add more datatypes to a control.

Table 47. Extensible methods for adding datatypes

Java	.Net
Hashtable getTestDataTypes()	Hashtable GetTestDataTypes()
ITestData getTestData(String)	ITestData GetTestData(string testDataType)

Property verification

You can use the methods that are listed in Table 48: Extensible methods for adding properties on page 1137 to add more properties to a control.

Table 48. Extensible methods for adding properties

Java	.Net
Hashtable gerProperties()	Hashtable GerProperties()
object getProperty(String)	object GetProperty(string)

Data driving

You can use the methods that are listed in Table 49: Extensible methods for data driving on page 1137 to specify the method that should be used for data driving.

Table 49. Extensible methods for data driving

Java	.Net
MethodSpecification getDataDrivableCommand	MethodSpecification GetDataDrivableCommand

Reflection Support

You can use the reflection support that Java™ and .Net provide from the test script. With APIs such as getMethods() and Invoke(), you can access information about methods on the underlying object to use in remote invocation. This information includes method name and signature. You can use the methods that are listed in Table 50: Extensible methods for reflection support on page 1138 for reflection support.

Table 50. Extensible methods for reflection support

Java	.Net
getMethods()	GetMethods()
invoke()	Invoke()
getNonValueProper-ties()	

The getNonValueProperties() method returns the properties that are not supported because the value is a reference to an object that can not be serialized across process boundaries. These properties can still be accessed using the getProperty() methods. If such a property is returned to the test script, it is returned as a TestObject containing a reference to the object in the AUT. The test script must call the TestObject method unregister() to release the object.

Java domain proxy hierarchy

There are three types of Java™ controls. The Abstract Window Toolkit (AWT), Swing or Java Foundation Class (JFC), and Standard Widget Toolkit (SWT), or Eclipse controls. Rational® Functional TesterHCL OneTest™ UI supports testing all three types of controls under Java UI frameworks. For the complete list of Java controls that are mapped to individual proxies, see the rational_ft.rftcust file in C:\Program Files\IBM\SDP\FunctionalTester\bin C:\Program Files \HCL\HCLOneTest\FunctionalTester\bin. This file can help you understand of what proxy to inherit to create a new proxy for a control under the Java domain.

You can extend these key base proxies for the Java domain:

- JavaProxy
- JavaGuiProxy
- ComponentProxy and JComponentProxy
- JfcGraphicalSubItemProxy and ScrollableSwtGraphicalSubItemProxy

JavaProxy

The JavaProxy proxy is the base proxy for all Java domain proxies. Fundamental Java object operation methods such as getProperties() and getMethods() are all implemented in this proxy.

JavaGuiProxy

The JavaGuiProxy proxy is the base proxy for all Java user interface (UI) elements. This proxy implements the IGraphical interface, which has methods for performing UI actions, such as click, double-click, drag, and recording methods.

ComponentProxy and JComponentProxy

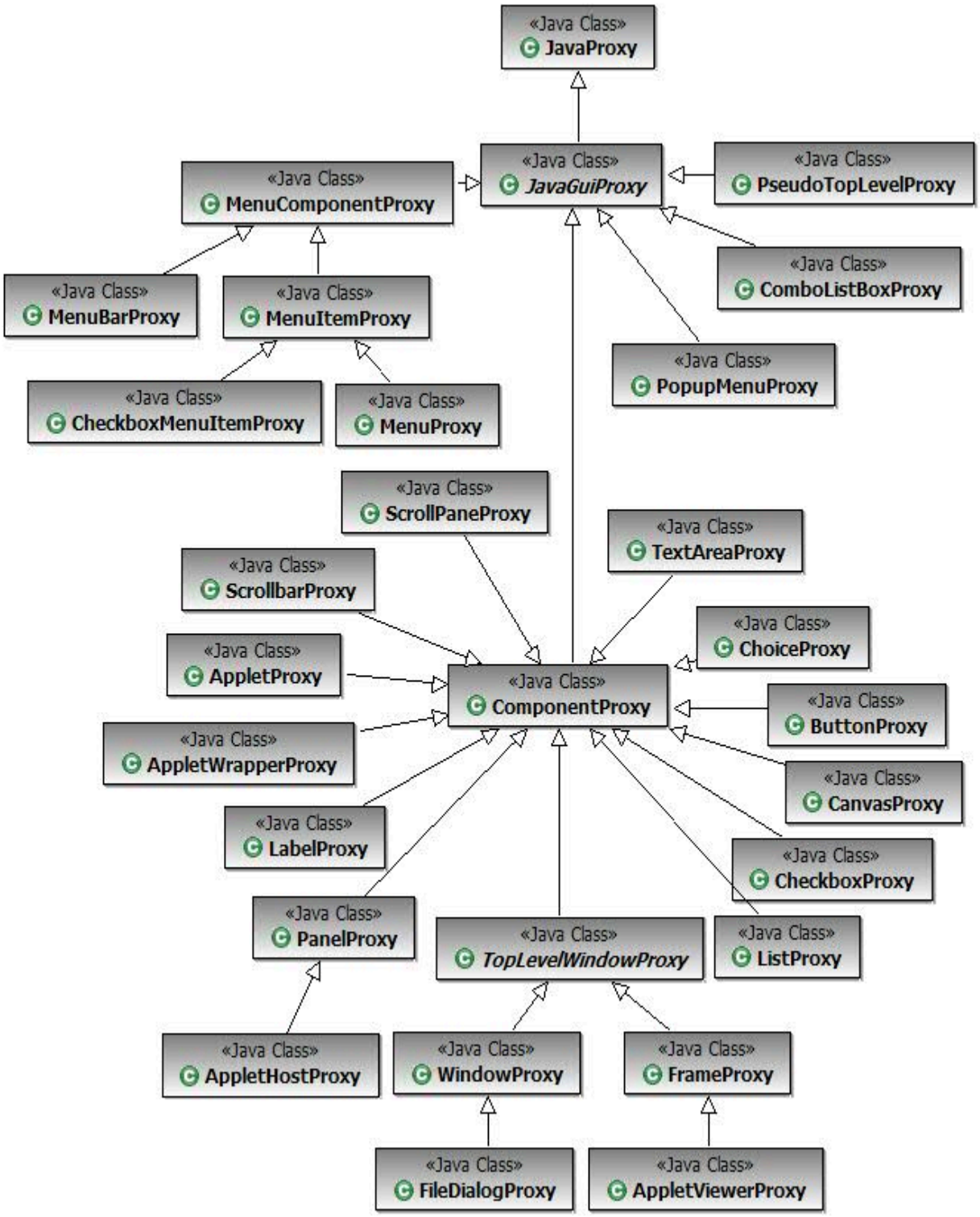
The ComponetProxy proxy is implemented for AWT components and JComponentProxy for JFC at the same level of the hierarchy. Methods such as getChilderen(), getParent(), getOwner(), getOwnedObjects(), and getMethods() are implemented in these proxies specific to the components.

JfcGraphicalSubItemProxy and ScrollableSwtGraphicalSubItemProxy

The JfcGraphicalSubItemProxy proxy is implemented for JFC and ScrollableSwtGraphicalSubItemProxy for SWT at the same level of the hierarchy. They provide recording and playback of methods with SubItems. When a control contains different parts then the proxy for that control can be inherited from this proxy.

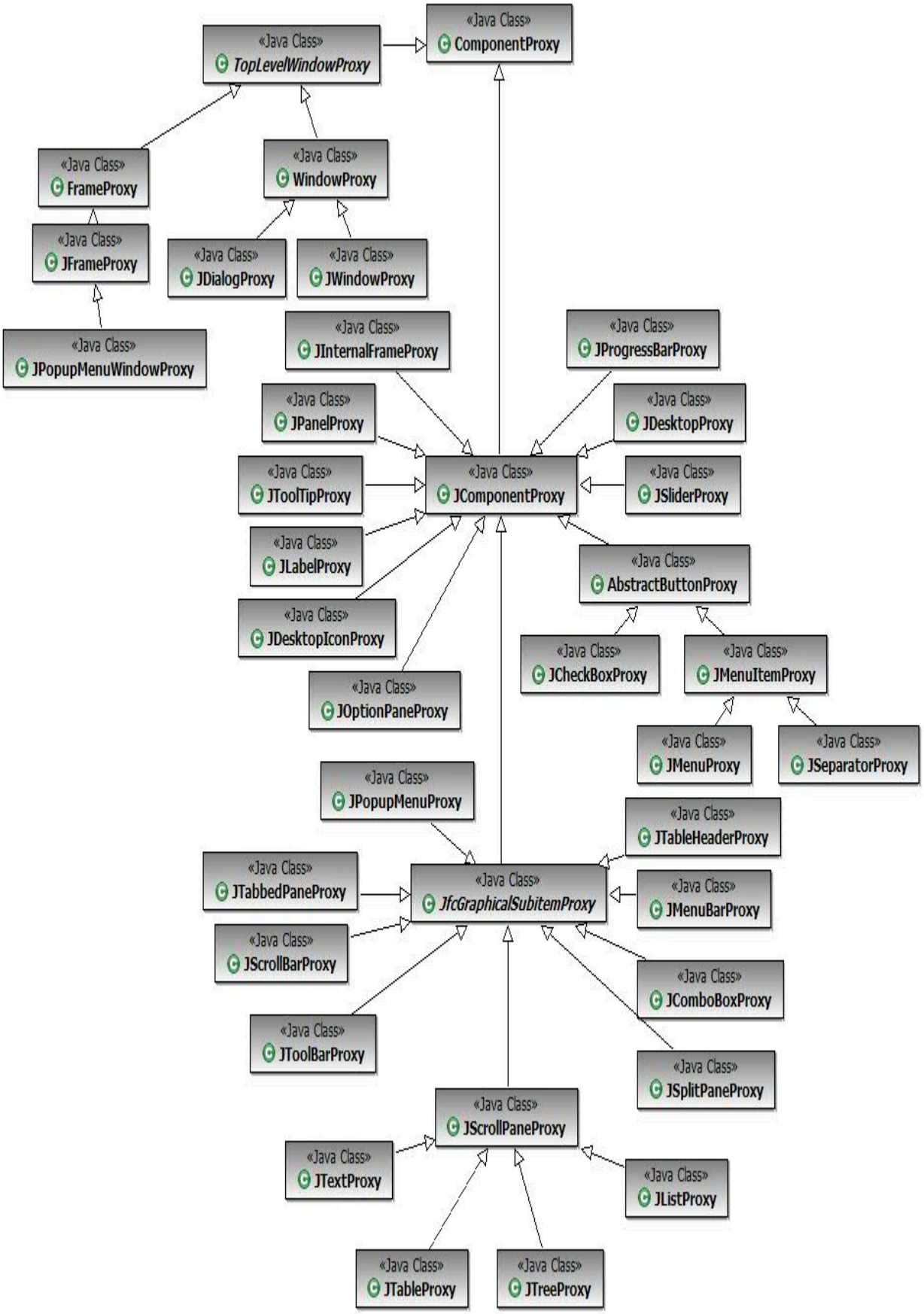
Proxy hierarchy for AWT controls

The following figure is a class diagram of the proxy hierarchy for AWT controls:



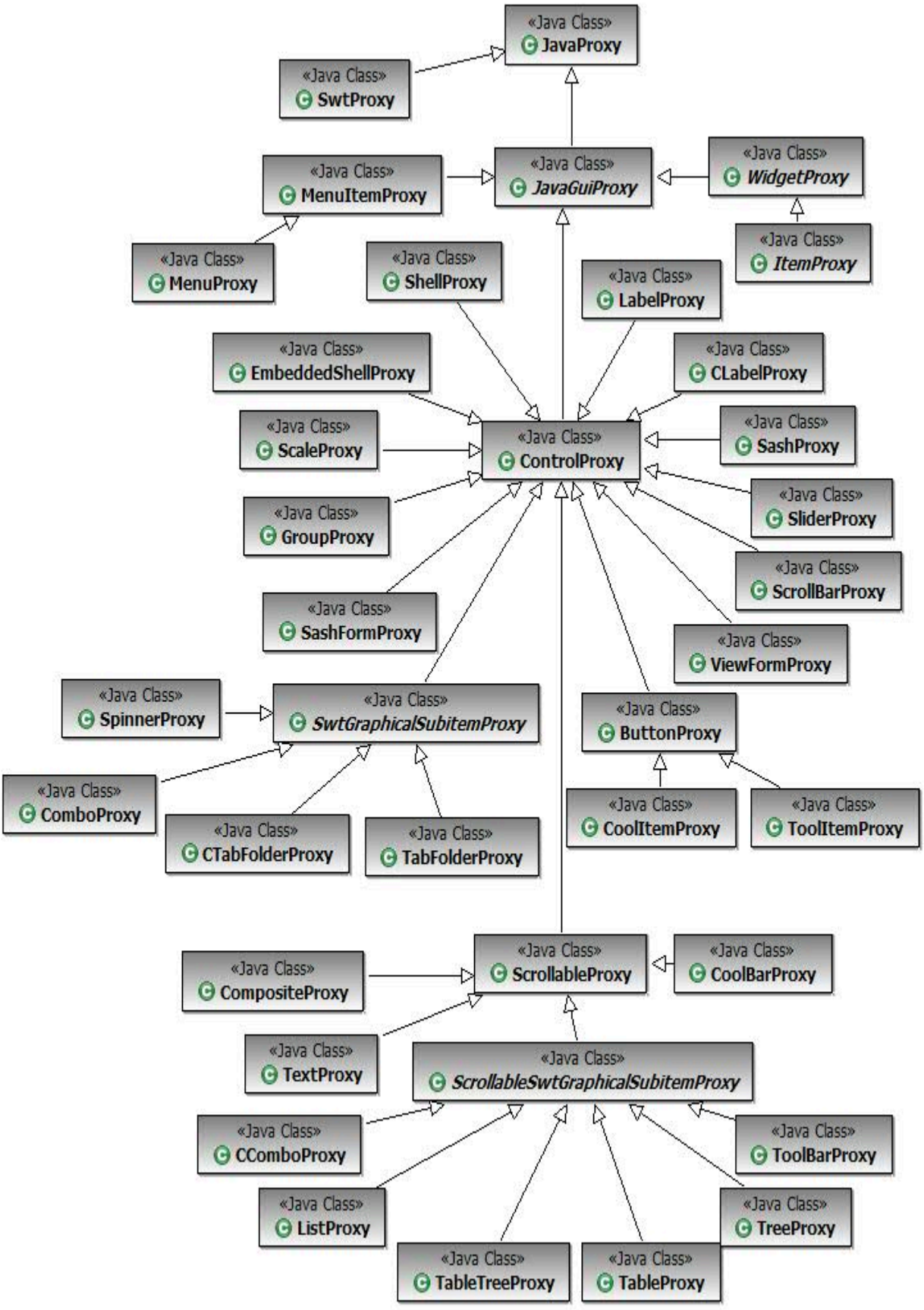
Proxy hierarchy for Swing or JFC controls

The following figure is a class diagram of the proxy hierarchy for Swing or JFC controls:


«Java Class»
TopLevelWindowProxy
«Java Class»
ComponentProxy
«Java Class»
FrameProxy
«Java Class»
WindowProxy
«Java Class»
JFrameProxy
«Java Class»
JDialogProxy
«Java Class»
JWindowProxy
«Java Class»
JPopupMenuWindowProxy
«Java Class»
JInternalFrameProxy
«Java Class»
JProgressBarProxy
«Java Class»
JPanelProxy
«Java Class»
JDesktopProxy
«Java Class»
JToolTipProxy
«Java Class»
JComponentProxy
«Java Class»
JSliderProxy
«Java Class»
JLabelProxy
«Java Class»
AbstractButtonProxy
«Java Class»
JDesktopIconProxy
«Java Class»
JCheckBoxProxy
«Java Class»
JMenuItemProxy
«Java Class»
JOptionPaneProxy
«Java Class»
JMenuProxy
«Java Class»
JSeparatorProxy
«Java Class»
JPopupMenuProxy
«Java Class»
JTableHeaderProxy
«Java Class»
JTabbedPaneProxy
«Java Class»
JfcGraphicalSubitemProxy
«Java Class»
JMenuBarProxy
«Java Class»
JScrollBarProxy
«Java Class»
JComboBoxProxy
«Java Class»
JToolBarProxy
«Java Class»
JSplitPaneProxy
«Java Class»
JScrollPaneProxy
«Java Class»
JTextProxy
«Java Class»
JListProxy
«Java Class»
JTableProxy
«Java Class»
JTreeProxy

Proxy hierarchy for SWT controls

The following figure is a class diagram of the proxy hierarchy for SWT controls:


«Java Class»
JavaProxy
«Java Class»
SwtProxy
«Java Class»
MenuItemProxy
«Java Class»
JavaGuiProxy
«Java Class»
WidgetProxy
«Java Class»
ItemProxy
«Java Class»
MenuProxy
«Java Class»
ShellProxy
«Java Class»
LabelProxy
«Java Class»
EmbeddedShellProxy
«Java Class»
CLabelProxy
«Java Class»
ScaleProxy
«Java Class»
ControlProxy
«Java Class»
SashProxy
«Java Class»
GroupProxy
«Java Class»
SliderProxy
«Java Class»
ScrollBarProxy
«Java Class»
SashFormProxy
«Java Class»
ViewFormProxy
«Java Class»
SpinnerProxy
«Java Class»
SwtGraphicalSubitemProxy
«Java Class»
ButtonProxy
«Java Class»
ComboProxy
«Java Class»
CTabFolderProxy
«Java Class»
TabFolderProxy
«Java Class»
CoolItemProxy
«Java Class»
ToolItemProxy
«Java Class»
CompositeProxy
«Java Class»
ScrollableProxy
«Java Class»
CoolBarProxy
«Java Class»
TextProxy
«Java Class»
ScrollableSwtGraphicalSubitemProxy
«Java Class»
CComboProxy
«Java Class»
ToolBarProxy
«Java Class»
ListProxy
«Java Class»
TreeProxy
«Java Class»
TableTreeProxy
«Java Class»
TableProxy

.Net domain proxy hierarchy

There are four key base proxies that you can extend to create new proxies for a control in the .Net domain. They are ObjectProxy, ComponentProxy, ControlProxy, and ControlWithSubobjectsProxy.

ObjectProxy

The ObjectProxy proxy is the base proxy for .Net domain proxies. By default it is mapped to System.Object class objects. It contains the default implementation of various base class methods, such as GetChildren(), GetMappableChildren(), and GetRecognitionProperties(). This is a non-UI proxy. If your control is derived from the System.Object class, then you can inherit the proxy for the control from ObjectProxy.

ComponentProxy

The ComponentProxy proxy is the base class for dealing with objects derived from System.ComponentModel.Component. This proxy implements TestObject methods such as GetParent() and GetMappableChildren().

ControlProxy

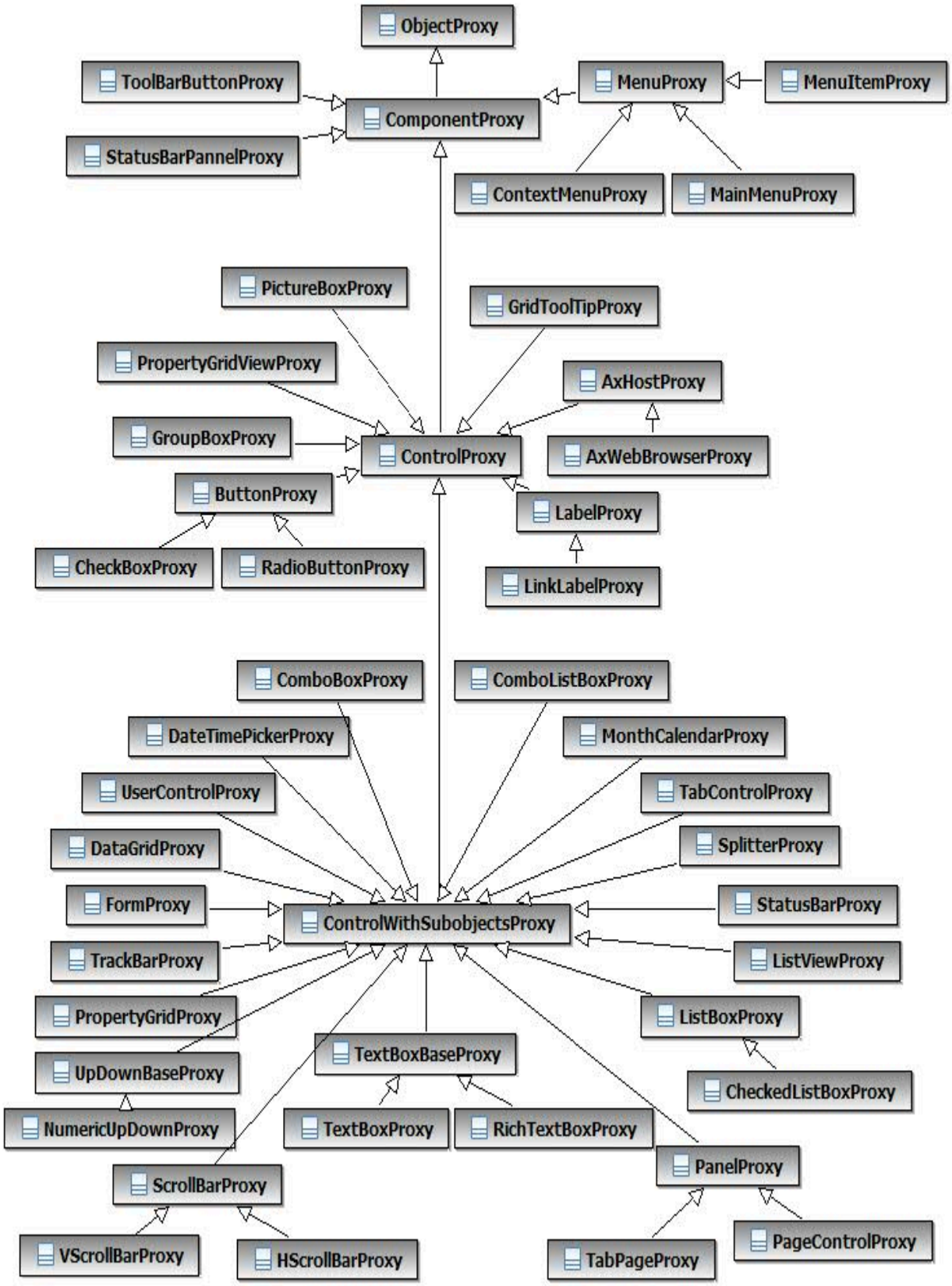
By default, any .Net control that you derive from the System.Windows.Forms class is mapped to the ControlProxy proxy. It provides both coordinates-based record and playback support for mouse actions such as click, drag, and hover. It also provides support for properties verification point, scrolling object, and object related point into view.

ControlWithSubobjectsProxy

The ControlWithSubobjectsProxy proxy implements basic functionalities for container controls such as Form and UserControl. This proxy also provides support for controls with parts that can be clicked and addressed, for example DataGrid control that has SubItem Cell and header. In addition, the ControlWithSubobjectsProxy provides support for record and playback for scrollbar SubItem, scrolling for SubItem in the control and SubItem in nested ScrollablControl parent.

Proxy hierarchy for .Net controls

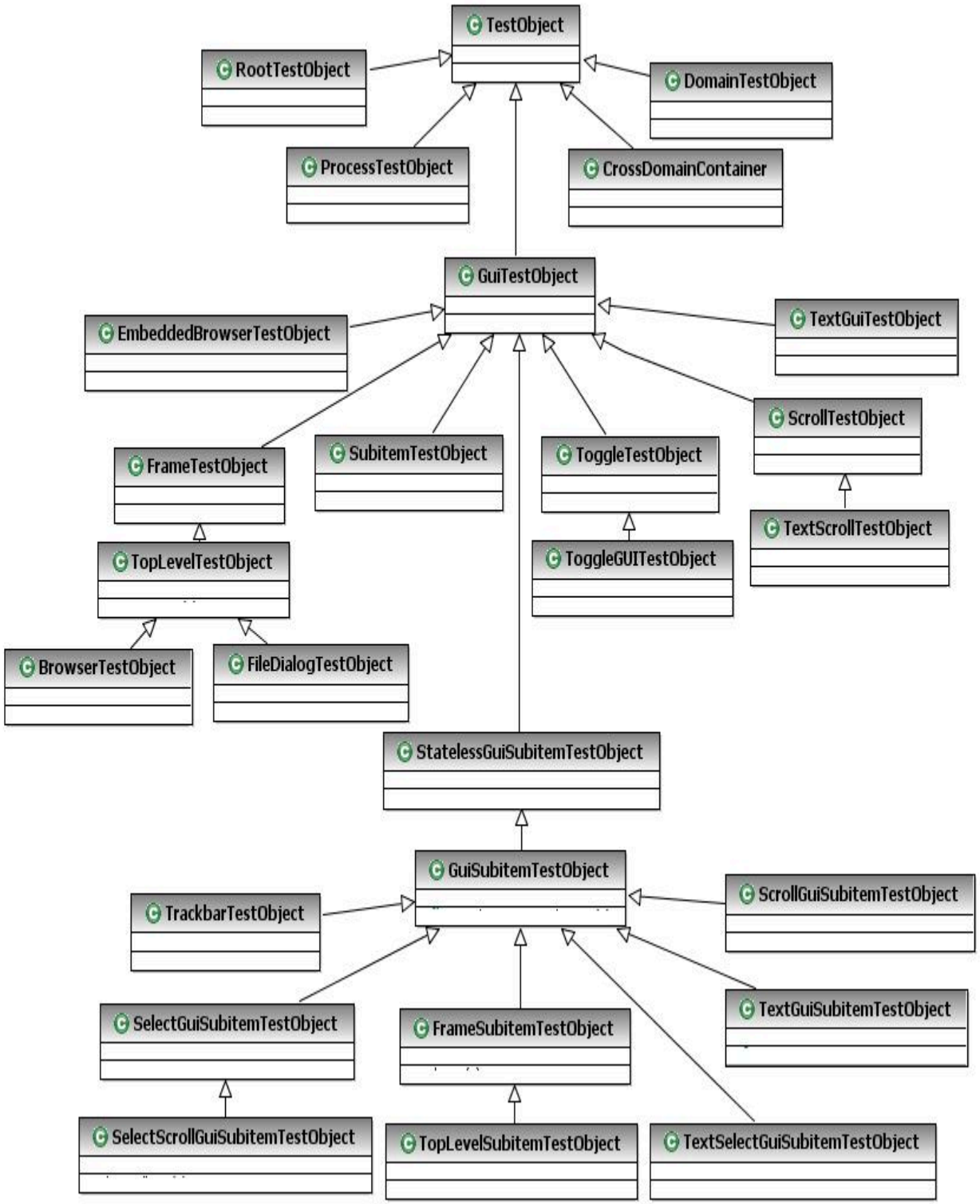
The following figure is a class diagram of the proxy hierarchy for .Net controls:


ObjectProxy
ToolBarButtonProxy
MenuProxy
MenuItemProxy
ComponentProxy
StatusBarPannelProxy
ContextMenuProxy
MainMenuProxy
PictureBoxProxy
GridToolTipProxy
PropertyGridViewProxy
AxHostProxy
GroupBoxProxy
ControlProxy
AxWebBrowserProxy
ButtonProxy
LabelProxy
CheckBoxProxy
RadioButtonProxy
LinkLabelProxy
ComboBoxProxy
ComboListBoxProxy
DateTimePickerProxy
MonthCalendarProxy
UserControlProxy
TabControlProxy
DataGridProxy
SplitterProxy
FormProxy
ControlWithSubobjectsProxy
StatusBarProxy
TrackBarProxy
ListViewProxy
PropertyGridProxy
ListBoxProxy
TextBoxBaseProxy
UpDownBaseProxy
CheckedListBoxProxy
NumericUpDownProxy
TextBoxProxy
RichTextBoxProxy
PanelProxy
ScrollBarProxy
VScrollBarProxy
HScrollBarProxy
TabPageProxy
PageControlProxy

TestObject class diagram and canonical names

Rational® Functional TesterHCL OneTest™ UI has a predefined set of TestObjects that are hierarchically grouped and each TestObject has a set of predefined methods.

The following class diagram shows all the TestObjects that are available with Rational® Functional TesterHCL OneTest™ UI and their relationships.



Canonical names

Canonical names are short string names for fully qualified TestObject class names. If you want to change the TestObject that is mapped to a proxy, you must override the `getTestObejctClassName()` API to return the

canonical name that the customization file specifies. The complete list of canonical names for TestObjects that Rational® Functional TesterHCL OneTest™ UI defines are declared as member variables to the `com.rational.test.ft.domain.ProxyTestObject` class for Java™ and the `Rational.Test.Ft.Domain.ProxyTestObject` class for .NET.

Canonical member variables

The ProxyTestObject for both Java and .Net implementations define the following canonical member variables:

```
BROWSERTESTOBJECT_CLASSNAME = "BrowserTestObject";
DOCUMENTTESTOBJECT_CLASSNAME = "DocumentTestObject";
DOMAINTESTOBJECT_CLASSNAME = "DomainTestObject";
FILEDIALOGTESTOBJECT_CLASSNAME = "FileDialogTestObject";
FRAMETESTOBJECT_CLASSNAME = "FrameTestObject";
GUISUBITEMTESTOBJECT_CLASSNAME = "GuiSubitemTestObject";
GUITESTOBJECT_CLASSNAME = "GuiTestObject";
INTERNALFRAMETESTOBJECT_CLASSNAME = "InternalFrameTestObject";
PROCESSTESTOBJECT_CLASSNAME = "ProcessTestObject";
SCROLLTESTOBJECT_CLASSNAME = "ScrollTestObject";
SCROLLSUBITEMTESTOBJECT_CLASSNAME = "ScrollGuiSubitemTestObject";
STATELESSGUISUBITEMTESTOBJECT_CLASSNAME = "StatelessGuiSubitemTestObject";
SUBITEMTESTOBJECT_CLASSNAME = "SubitemTestObject";
TESTOBJECT_CLASSNAME = "TestObject";
TEXTGUITESTOBJECT_CLASSNAME = "TextGuiTestObject";
TEXTGUISUBITEMTESTOBJECT_CLASSNAME = "TextGuiSubitemTestObject";
TEXTSCROLLTESTOBJECT_CLASSNAME = "TextScrollTestObject";
TEXTSELECTGUISUBITEMTESTOBJECT_CLASSNAME = "TextSelectGuiSubitemTestObject";
SELECTGUISUBITEMTESTOBJECT_CLASSNAME = "SelectGuiSubitemTestObject";
SELECTSCROLLGUISUBITEMTESTOBJECT_CLASSNAME = "SelectScrollGuiSubitemTestObject";
TOGGLEGUITESTOBJECT_CLASSNAME = "ToggleGUITestObject";
TOGGLETESTOBJECT_CLASSNAME = "ToggleTestObject";
TOPLEVELTESTOBJECT_CLASSNAME = "TopLevelTestObject";
TOPLEVELSUBITEMTESTOBJECT_CLASSNAME = "TopLevelSubitemTestObject";
TRACKBARTESTOBJECT_CLASSNAME = "TrackbarTestObject";
CROSSDOMAINCONTAINER_CLASSNAME = "CrossDomainContainer";
EMBEDDEDBROWSERTESTOBJECT_CLASSNAME = "EmbeddedBrowserTestObject";
FRAMESUBITEMTESTOBJECT_CLASSNAME = "FrameSubitemTestObject";
```

SubItems

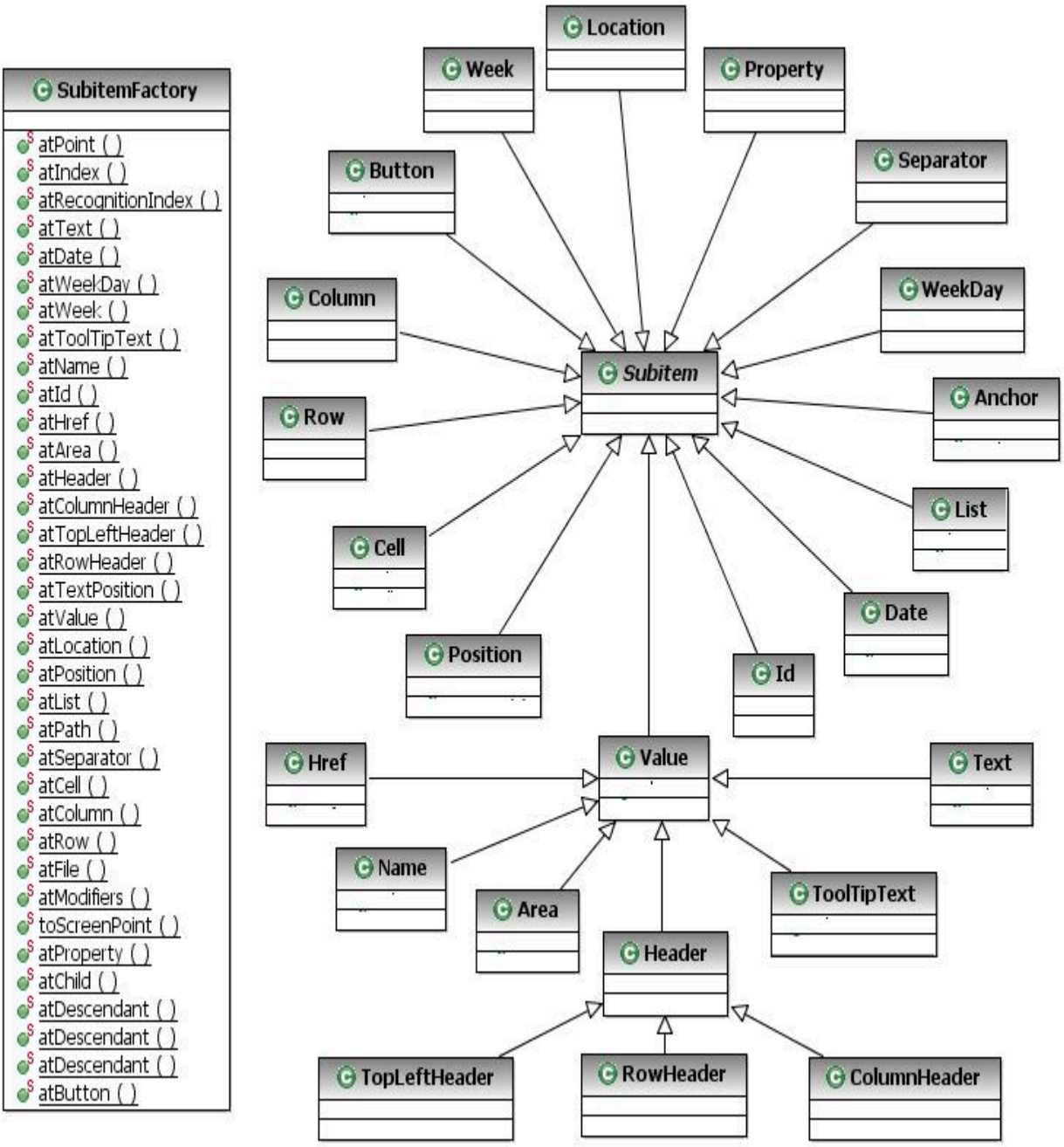
SubItems are part of TestObjects. The ObjectMap does not contain the SubItems, because they are not mapped.

Some examples of SubItems follow:

- Menu object has MenuItem as Subitem
- List object has ListItem as Subitem
- Tree object has TreeNode as Subitem
- Table object has cell, row and column as Subitem

Class diagram

The following class diagram shows all the SubItems and their relationships that Rational® Functional TesterHCL OneTest™ UI provides.



List of SubItems

Table Table 51: SubItems on page 1150 lists all SubItems and their descriptions:

Table 51. SubItems

SubItem	Description
Area(*a*)	Specifies an independent clickable area within an HTML Image map. The nested SubItem (a) can be a point or an Index.
Cell(*c, r*)	Specifies a cell within a table. The nested SubItems (c and r) must be a column and a row respectively.
Column(*s*)	Specifies a column within a table by the text of a row label (s).
Column(n), Column(*k, v*)	Specifies a column within a table by index.
Column(*k1, v1, k2, v2*)	Specifies a column within a table by key value pair or pairs. This SubItem supports up to three pairs.
File(*d, f*)	Specifies a file within a directory that the AWT file dialog proxy uses.
Header(*c*)	Specifies a header within a table. The nested SubItem (h) must be a column.
Href(*s*)	Matches the first SubItem that has the specified HREF within the object or SubItem.
Id(*n*)	Matches the first SubItem that has the specified ID within the object or SubItem.
Index(*n*)	The nth SubItem within an object or SubItem.
List(...)	An ordered sequence of SubItems. Each item indicates a further SubItem within the previous SubItem.
Location(*s*)	Matches the first SubItem that has the specified named location within the object or SubItem. A location name must be unique within the parent.
Name(*s*)	Matches the first SubItem that has the specified name within the object or SubItem.
Name(*s, n*)	Matches the nth SubItem that has the specified name within the object or SubItem. The count (n) is based on zero (0).
Path(*s*)	A string encoding of a List that is used to make the script more readable. The SubItems in the list must be Text or Index and the final SubItem can optionally be one of several locations. The index is encoded as atIndex(n) and each item in the encoded list is separated by the characters `->`. A path is transformed to a list and the proxies are never aware of these.
Point(*x, y*)	Relative coordinates within the object or SubItem.
Position(*n*)	Specifies a position on a SubItem, typically a splitter or frame.
Row(*s*)	Specifies a row within a table by the text of a column header or headers.

Table 51. SubItems (continued)

SubItem	Description
Row(*n*), Row(*k, v*)	Specifies a row within a table by index.
Row(*k1, v1, k2, v2*)	Specifies a row within a table by key value pair or pairs. This SubItem supports up to three pairs
Separator(*n*)	Matches the nth separator in a menu or toolbar. The count (*n*) is 0-based.
Text(*s*)	Matches the first SubItem that has the specified text within the object or SubItem.
Text(*s, n*)	Matches the nth SubItem that has the specified text within the object or SubItem. The count (*n*) is 0-based.
TextPosition(*n*)	Matches the first SubItem that has the specified text position within the object or SubItem.
Value(*v*)	Matches the first SubItem that has the specified value within the object or SubItem.

SubItem values

The following table contains the list of SubItem values which are defined as members to `com.rational.test.ft.script.Location` for Java and `Rational.Test.Ft.Script.Location` for .NET.

Table 52. SubItem values

Constant	Value
ARROW	"ARROW"
BACK_BUTTON	"BACK_BUTTON"
BACKGROUND	"BACKGROUND"
BOTTOM_EDGE	"BOTTOM_EDGE"
CAPTION	"CAPTION"
CHECKBOX	"CHECKBOX"
CLOSE_BUTTON	"CLOSE_BUTTON"
CONTEXTHELP_BUTTON	"CONTEXTHELP_BUTTON"
DROPDOWN	"DROPDOWN"
IME_BUTTON	"IME_BUTTON"
LEFT_EDGE	"LEFT_EDGE"
MAXIMIZE_BUTTON	"MAXIMIZE_BUTTON"

Table 52. SubItem values (continued)

Constant	Value
MINIMIZE_BUTTON	"MINIMIZE_BUTTON"
MONTH	"MONTH"
PARENTROWS	"PARENTROWS"
PLUS_MINUS	"PLUS_MINUS"
POPUP	"POPUP"
RIGHT_EDGE	"RIGHT_EDGE"
SCROLL_DOWN	"SCROLL_DOWN"
SCROLL_DOWNBUTTON	"SCROLL_DOWNBUTTON"
SCROLL_ELEVATOR	"SCROLL_VERTICAL_ELEVATOR"
SCROLL_HORIZONTAL_ELEVATOR	"SCROLL_HORIZONTAL_ELEVATOR"
SCROLL_LEFT	"SCROLL_LEFT"
SCROLL_LEFTBUTTON	"SCROLL_LEFTBUTTON"
SCROLL_MAXBUTTON	"SCROLL_MAXBUTTON"
SCROLL_MINBUTTON	"SCROLL_MINBUTTON"
SCROLL_RIGHT	"SCROLL_RIGHT"
SCROLL_RIGHTBUTTON	"SCROLL_RIGHTBUTTON"
SCROLL_UP	"SCROLL_UP"
SCROLL_UPBUTTON	"SCROLL_UPBUTTON"
SCROLL_VERTICAL_ELEVATOR	"SCROLL_VERTICAL_ELEVATOR"
SHOWHIDE_BUTTON	"SHOWHIDE_BUTTON"
SYSTEM_MENU	"SYSTEM_MENU"
THUMB	"THUMB"
TODAY	"TODAY"
TOP_EDGE	"TOP_EDGE"
YEAR	"YEAR"

Value classes and value managers

The following code examples show several value classes and value managers.

Value classes

A value class is a Java™ or .Net class containing data that is useful to interact with. An instance of a value class can persist and can be compared to other instances of the same class. This is a basic capability of all value classes.

This example code shows a Java value class:

```
package sdk.sample.value;

public class SimpleValue
{
 String data = null;
 public SimpleValue(String data)
 {
  this.data = data;
 }
 public String getValue()
 {
  return this.data;
 }
 public String toString()
 {
  return "SimpleValue("+data+")";
 }
}
```

This example code shows a .Net value class:

```
using System;

namespace SDK.Sample.Value
{

 public class SimpleValue
 {
  private String data = null;

  public SimpleValue(String data)
  {
   this.data = data;
  }
  public String GetValue()
  {
   return this.data;
  }
  public override String ToString()
  {
   return "SimpleValue("+data+")";
  }
 }

}
```

Value Managers

Value managers interact with value classes so that value class objects can be serialized and compared and made to persist. Value manager classes can be dynamically added to the set of supported managers. After a new manager has been registered, any property of the newly supported value class is automatically expressed in the set of properties associated with a test object.

This example code shows a Java value manager:

```
package sdk.sample.value;

import com.rational.test.ft.value.managers.*;

public class SimpleValueManager implements IManageValueClass, IStringTableLookup
{
 private static final String CLASSNAME = "sdk.sample.value.SimpleValue";
 private static final String CANONICALNAME = ".simple_value";

 private static final String DATA = "Data";


         public void persistOut(Object theObject, IPersistOut persist,
         IAuxiliaryDataManager auxData)
 {
  SimpleValue simple = (SimpleValue)theObject;
  persist.write(DATA, simple.getValue());
 }

 public Object persistIn(IPersistIn persist,
       IAuxiliaryDataManager auxData)
 {
  String data = (String)persist.read(0);
  return new SimpleValue(data);
 }

 public Object persistIn(IPersistInNamed persist,
       IAuxiliaryDataManager auxData)
 {
  String data = (String)persist.read(DATA);
  return new SimpleValue(data);
 }

 public int compare(Object left, Object right, ICompareValueClass nested)
 {
  if ( left == null || right == null )
   return ( left == right ? 100 : 0 );
  if ( !(right instanceof SimpleValue) ) return 0;
  SimpleValue l = (SimpleValue)left;
  SimpleValue r = (SimpleValue)right;
  return ( l.equals(r) ? 100 : 0 );
 }


         public Object createValue(Object sourceToCopy)
 {
  if ( sourceToCopy instanceof SimpleValue )
```

```
   return new SimpleValue(((SimpleValue)sourceToCopy).getValue());
  return null;
 }

 public String getCanonicalName()
 {
  return CANONICALNAME;
 }


 public String getClassName()
 {
  return CLASSNAME;
 }

 public String doLookup(Object lookup)
 {
  String retVal = null;
  if (lookup instanceof SimpleValue && lookup != null)
  {
   retVal = com.rational.test.ft.services.StringTableService.getString(
    ((SimpleValue)lookup).getValue());
   // If they are the same return null so we won't bother changing VP data, etc.
   if (retVal == ((SimpleValue)lookup).getValue())
   {
    retVal = null;
   }
  }
  return retVal;
 }

}
```

This example code shows a .Net value manager:

```
using System;
using Rational.Test.Ft.Value.Managers;

namespace SDK.Sample.Value
{
 public class SimpleValueManager: IManageValueClass
 {
  private const System.String CLASSNAME = "SDK.Sample.Value.SimpleValue";
  private const System.String CANONICALNAME = ".simpe_value";

  private const System.String DATA = "Data";

  public virtual void  PersistOut(System.Object theObject, IPersistOut persist, IAuxiliaryDataManager
 auxData)
  {
   SimpleValue simple = (SimpleValue)theObject;
   persist.Write(DATA, simple.GetValue());
  }

  public virtual System.Object PersistIn(IPersistIn persist, IAuxiliaryDataManager auxData)
  {
   String data = (String)persist.Read(0);
```

```
   return new SimpleValue(data);
  }

  public virtual System.Object PersistIn(IPersistInNamed persist, IAuxiliaryDataManager auxData)
  {
   String data = (String)persist.Read(DATA);
   return new SimpleValue(data);
  }

  public virtual int Compare(System.Object left, System.Object right, ICompareValueClass nested)
  {
   if ( left == null || right == null )
    return ( left == right ? 100 : 0 );
   if ( !(right is SimpleValue) ) return 0;

   SimpleValue l = (SimpleValue)left;
   SimpleValue r = (SimpleValue)right;
   return ( l.Equals(r) ? 100 : 0 );
  }


  public virtual System.Object CreateValue(System.Object sourceToCopy)
  {
   if ( sourceToCopy is SimpleValue )
    return new SimpleValue(((SimpleValue)sourceToCopy).GetValue());
   return null;
  }

  public virtual System.String GetCanonicalName()
  {
   return CANONICALNAME;
  }

    public virtual System.String GetClassName()
  {
   return CLASSNAME;
  }
 }
}
```

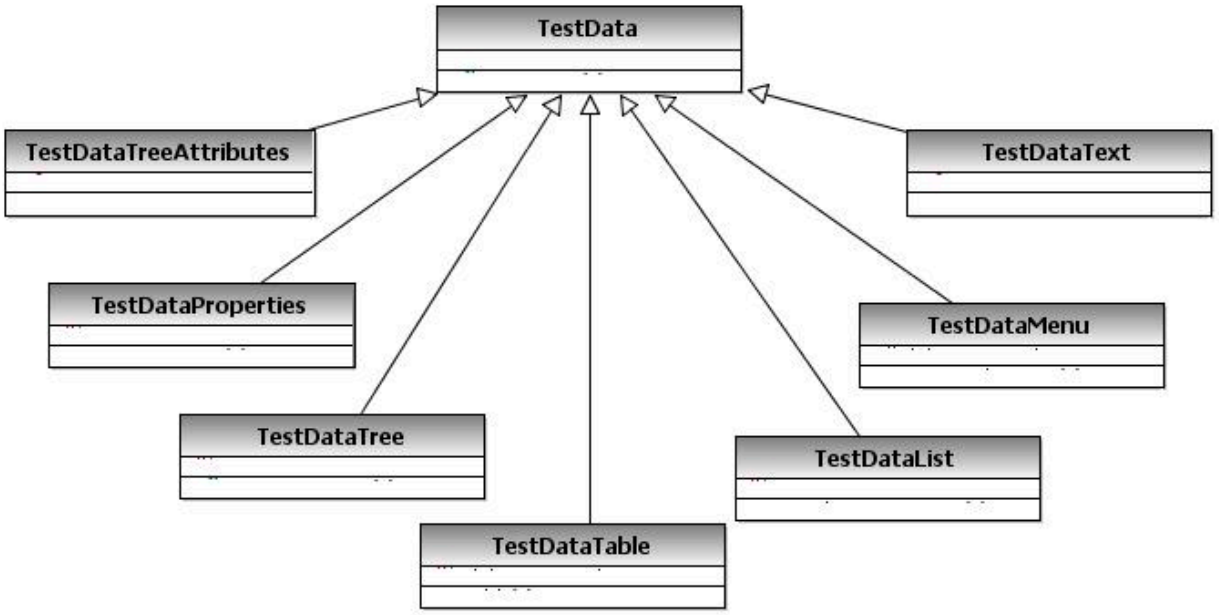
TestData types

The Rational® Functional TesterHCL OneTest™ UI framework calls the `getTestDataTypes()` and `getTestData(String)` proxy methods for extracting data from controls for verification and reference. These methods are used during the creation and play back of data verification points. You can override the `getTestDataTypes()` method to add more specific data types for a control.

For example, a TextBox control can have **text** and **selected-text** as the supported data types. Each of these types is associated with a string name and description that the proxy defines. This name is passed to the `getTestData(String)` API for getting the control data. While implementing the `getTestData(String)` API, you must use the appropriate predefined data types and populate them with the control data and return them accordingly.

Class diagram

The following class diagram shows the predefined test data types that Rational® Functional TesterHCL OneTest™ UI makes available:



The following predefined data types are some of the data types that you can use while implementing the getTestData() proxy API

TestDataText

The `TestDataText` type represents a string value.

This example code shows how to implement the TestDataText data type in Java™:

```
import com.rational.test.ft.vp.ITestData;
import com.rational.test.ft.vp.impl.TestDataText;

ITestData testData =  null;
testData = new TestDataText( getSelectedText());
return testData ;
```

This example code shows how to implement the TestDataText data type in .Net:

```
Rational.Test.Ft.Vp.ITestData testData = null ;
object item = ((ComboBox)theTestObject).SelectedItem ;
testData = new Rational.Test.Ft.Vp.Impl.TestDataText(((ComboBox)theTestObject).GetItemText(item));
return testData;
```

TestDataList

The `TestDataList` type represents a list of items, for example items in a ListBox and a single column of a table.

This example code shows how to implement the TestDataList data type in Java:

```
import com.rational.test.ft.vp.ITestData;
import com.rational.test.ft.vp.impl.TestDataElementList;
import com.rational.test.ft.vp.impl.TestDataList;

Object[] items = getListItemObjects();
TestDataElementList testData = new TestDataElementList();
for ( int i = 0; i < items.length; i ++  )
{
 if ( items[i] != null )
 {
  testData.add(new TestDataElement(items[i], false));
  nonNullValueExist = true;
 }
 else
  testData.add(null);
}
return (new TestDataList(testData));
```

This example code shows how to implement the TestDataList data type in .Net:

```
Rational.Test.Ft.Vp.ITestData testData = null ;
string[] itemList = new string[((ComboBox)theTestObject).Items.Count] ;
for(int i=0; i < ((ComboBox)theTestObject).Items.Count; i++)
{
 object item = ((ComboBox)theTestObject).Items[i] ;

 if (item is string)
  itemList[i] = (string) item ;
 else
  itemList[i] = ((ComboBox)theTestObject).GetItemText(item) ;

}
testData = new Rational.Test.Ft.Vp.Impl.TestDataList(new
 Rational.Test.Ft.Vp.Impl.TestDataElementList(itemList)) ;
return testData;
```

TestDataTable

The `TestDataTable` type represents two-dimensional data that is contained in controls such as tables or grids.

This example code shows how to implement the TestDataTable data type in Java:

```
import com.rational.test.ft.vp.ITestData;
import com.rational.test.ft.vp.ITestDataTable;
import com.rational.test.ft.vp.impl.TestDataTable;
import com.rational.test.ft.vp.impl.TestDataTableRegion;
.
.
int rowCount = getRowCount();
int colCount = getColumnCount();

object[] rowElements;
rowElements = new object[colCount];
```

```
for (int row = 0; row < rowCount; ++row)
{
 for (int col = 0; col < colCount; ++col)
 {
  object item = this.getItemText(row, col);
  if (item != null)
   rowElements[col] = item.ToString();
 }
 testData.add(rowElements);
}


for (int col = 0; col < colCount; ++col)
{
 object item = this.getColumnName(col);
 if (item != null)
  data.setColumnHeader(col, header);
}


testData.addComparisonRegion(TestDataTableRegion.allCells());
testData.setCompareBothByLeftRegions(true);

return testData;
```

This example code shows how to implement the TestDataTable data type in .Net:

```
Rational.Test.Ft.Vp.ITestData testData = null;
System.Data.DataTable dataTable = GetControlData();

int colCount = dataTable.Columns.Count;
int rowCount = dataTable.Rows.Count;

object[] rowElements;
rowElements = new object[colCount];

for (int row = 0; row < rowCount; ++row)
{
 for (int col = 0; col < colCount; ++col)
 {
  object item = null;
  item = dataTable.Rows[row][col];
  if (item != null)
   item = item.ToString();
  rowElements[col] = item;
 }
 testData.Add(rowElements);
}


for (int col = 0; col < colCount; ++col)
{
 string columnName = dataTable.Columns[col].ColumnName;
 if (columnName != null && !columnName.Equals(string.Empty))
  testData.SetColumnHeader(col, columnName);
}
```

```
testData.AddComparisonRegion(TestDataTableRegion.AllCells());
testData.SetCompareBothByLeftRegions(true);

return testData;
```

TestDataTree

The `TestDataTree` type represents a tree data structure.

This example code shows how to implement the TestDataTree data type in .Net:

```
public override Rational.Test.Ft.Vp.ITestData GetTestData(string testDataType)
{
.
.
 Rational.Test.Ft.Vp.ITestData testData = new TestDataTree(GetRootNodes());
 return testData;
.
.
}


private ITestDataTreeNodes GetRootNodes()
{
 System.Collections.ArrayList nodeCache = new System.Collections.ArrayList(80);
 System.Windows.Forms.TreeNodeCollection rootNodes = ((TreeView)this.theTestObject).Nodes;

 if ( rootNodes != null && rootNodes.Count> 0 )
 {
  for ( int i = 0; i < children.Length; ++i )
  {
   nodeCache.Add(GetNode(children[i], null));
  }
 }

 ITestDataTreeNode[] nodes = new TestDataTreeNode[nodeCache.Count];
 System.Array array = nodeCache.ToArray();
 for ( int i = 0; i < array.Length; ++i )
 {
  nodes[i] = (ITestDataTreeNode)array.GetValue(i);
 }
 TestDataTreeNodes testNodes = new TestDataTreeNodes(nodes);

 return testNodes;
}

// Gets called by GetRootNodes()

private ITestDataTreeNode GetNode(System.Object item, ITestDataTreeNode parent)
{
 String text = ((TreeNode)item).Text;
 ITestDataTreeAttributes attr = new TestDataTreeAttributes(text);
 ITestDataTreeNode node = new TestDataTreeNode(parent, text, null, false);

 System.Collections.ArrayList nodeCache = new System.Collections.ArrayList(20);
```

```
 System.Windows.Forms.TreeNodeCollection childrenNodes = ((TreeNode)item).Nodes;

 if ( childrenNodes != null && childrenNodes.Count> 0 )
 {
  int length = childrenNodes.Count;
  if ( length> 0 )
  {
   for ( int i = 0; i < length; ++i )
   {
    nodeCache.Add(GetNode(children[i], node));
   }
   int size = nodeCache.Count;
   if ( size> 0 )
   {
    ITestDataTreeNode[] childNodes = new ITestDataTreeNode[size];
    System.Array array = nodeCache.ToArray();
    for ( int i = 0;i < size; i ++ )
     childNodes[i] = (ITestDataTreeNode)array.GetValue(i);
    node.SetChildren(childNodes);
   }
  }
 }
 return node;
}
```

Proxy exceptions

Rational® Functional TesterHCL OneTest™ UI has a set of predefined exceptions that covers wide ranges of errors that are usually expected while functional testing. These exceptions are available for both the Java™ and .NET proxy development frameworks.

For best results, use the predefined exceptions while developing your ProxyObjects. The available exceptions and their hierarchies are as follows.

Table 53.

Exceptions
`AmbiguousRecognitionException`
`ApplicationFrameworkException`
`BadArgumentException`
`CookieNotFoundException`
`CoordinateOffScreenException`
`CoordinateOnWrongObjectException`
`CoordinateOnWrongSubitemException`
`datasetException`
`InvalidSignatureException`
`InvalidSubitemException`

Table 53. (continued)

Exceptions
`InvalidTestDataTypeException`
`InvalidTestObjectException`
`InvalidWindowHandleException`
`InvocationTargetException`
`MethodNotFoundException`
`NestedException`
`NoSuchRegistryKeyException`
`NotSupportedOnUnixException`
`NotYetAbleToPerformActionException`
`ObjectIsDisposedException`
`ObjectNotFoundException`
`ObjectNotInMapException`
`PropertyAccessException`
`PropertyNotFoundException`
`RationalTestException`
`RationalTestRemoteException`
`RecorderException`
`StringNotInCodePageException`
`SubitemNotFoundException`
`TargetGoneException`
`TestObjectMethodEventException`
`UnableToAccomplishAllHooksException`
`UnableToDeleteCookieException`
`UnableToHookException`
`UnableToPerformActionException`
`UnregisteredObjectException`
`UnsupportedActionException`
`UnsupportedAnchorException`
`UnsupportedMethodException`

Table 53. (continued)

Exceptions
`UnsupportedSubitemException`
`WindowActivateFailedException`
`WindowHandleNotFoundException`
`WrappedException`

Proxy development best practices

Following best practices while you develop proxies can help make your work more efficient and effective. Consider these methods to enhance your development efforts.

Using the appropriate hierarchy for GUI objects

GUI objects are arranged in two related hierarchies: parent-child and owner-owned. An example of parent-child relationship is a dialog box and a contained button. An example of an owner-owned relationship is a top-level window and a dialog box. Use any one type of hierarchy for an object.

In proxy implementation, it is common for the underlying object model (for example, Java™ and HWND) to confuse these two relationships, and treats the owner-owned relationship as an asymmetric parent-child relationship. In this case, the proxy must deny having a parent when the preferable relationship model is owner/owned. The methods that can be used for going through the hierarchy are getParent(), getChildren(), getOwner(), and getOwned().

Avoid returning different types for the same property

Properties are named values. The property itself does not have a type; the value has a type. Avoid making proxies return different types for the same property. Sometimes a property value can be a reference to an object rather than a value. If such a value is returned to a script, it is returned as a TestObject. The methods to access the properties include getProperty(), setProperty(), getProperties(), and getNonValueProperties().

Using the Object Library to assign recognition properties and weights for the objects

By default, the new proxy objects might not have object recognition properties and weights. Use the Object Library to assign the recognition properties and weights. The methods for accessing the object recognition properties and weights are getRecognitionProperties() and getRecognitionPropertyWeight(). If more than one object of the same class exists within the parent object, add the .classIndex property (a positive numeric value starting at 0), as a recognition property for the child object.

Managing mappable hierarchies

Generally, the entire hierarchy of the objects are mapped. However, certain objects are likely to change frequently between builds of the test application. For example, in Java, it is common to add panels to cluster objects together. With Rational® Functional TesterHCL OneTest™ UI the user can specify not to map the proxy object. Even though the non-mappable objects are not in the test object map, the object hierarchy lists them while you traverse

through the parent-child hierarchy. The methods for managing the mappable hierarchy include shouldBeMapped(), getMappableParent(), and getMappableChildren().

Using canonical properties

In some cases, the underlying object model supports a notion of properties, for example, Java, HTML, and .NET. Rational® Functional TesterHCL OneTest™ UI allow proxies to implement additional properties. If a proxy implements a property directly, the property name must have a different pattern to avoid confusion with any property of the object. Begin the regular properties name that might be used for recognition should with a dot (.). Certain administrative properties are used by the framework that cannot be used for recognition, these property names begin with a number sign (#).

Invoking object methods

In many cases, the underlying object has methods that can be found and invoked. These are usually managed directly by the framework and the domain implementation, but getMethod() is commonly implemented on a base proxy in a domain.

Using well-defined scriptable methods

Make mouse actions play back from the glass whenever possible. If a method name includes "click" or "drag", have mouse events perform the action . Do not use method names like "click" or "drag" if mouse events are not used to implement the action. Do not use method names that refer to keys or the keyboard if keyboard events do not implement the action. Make the methods reflect the action that is going to be performed and reliable during playback. Avoid heuristics.

Managing subitems

Objects can contain other objects and might have an internal structure that is not exposed as a nested object or objects. For example, a list might have items in it, but the items are not exposed by the list object as objects themselves. This kind of behavior is common in HWND-based objects. The most common method to deal with subitems are getSubitem() and the mouse action methods such as click(), drag(), and doubleClick(). The getSubitem() method returns a null or a string.

Use any of the following strategies if you do not want the objects in the subitems to be exposed as full-fledged objects:

- Declare the subitem as not mappable

  Declare the subitem proxy as not mappable. The object at point talks to its parent and the parent proxy is responsible for recording actions against the aggregate object and for supporting playback of subitem-based GUI actions.
- Make the subitem delegate the call to its parent

During recording the subitem proxy passes the processMouseEvent() calls to its parent. The parent proxy is responsible for recording actions against the aggregate object and for supporting playback of subitem-based GUI actions.

- Make the subitem delegate recording actions to its parent

  During recording the subitem proxy processes processMouseEvent() calls and generates the method calls on the parent object. The parent proxy is not responsible for recording actions against the aggregate object but is responsible for supporting playback of subitem-based GUI actions.

- Use child objects of subitems

  Have a new kind of object reference that is tagged so that the reference is unregistered after it is used as an anchor. This enables anchoring an object using subitems of another object. You can map the nested object as a child of the parent object.

Exceptions and Errors

Follow these rules while implementing exceptions and errors:

- Use public, documented exceptions

  Do not throw private or internal exceptions from the proxies. The API documentation for your proxies might not be available. Use standard exceptions, preferably RationalTestExceptions.

- Re-use exceptions

  Re-use the exceptions from the com.rational.test.ft package in Java or the Rational.Test.Ft namespace in .Net. If a java proxy throws a com.rational.test.ftMethodNotFoundException exception and the script is written in VB, the VB script gets a Rational.Test.Ft.MethodNotFoundException exception. If you throw an exception that does not have a name that begins with com under com.rational.test.ft it gets marshalled as a WrappedException exception.

- Use constructors

  All RationalTestExceptions that can be marshalled must support a constructor that takes a single string parameter.

- Make parallel implementations of .Net and Java exceptions

  If you add a new exception in a proxy implemented in Java that you expect to be able to be returned to the client, ensure you implement the same exception in .Net.

- Declare runtime exceptions

  In Java, exceptions must be declared , which might not be helpful for some testers. Many exceptions arise from almost any GUI method, so use runtime exceptions instead of exceptions in Java. Avoid using errors. An Error should cause the entire playback or recorder session to stop.

- Include ObjectNotFound exceptions

If the object is not found, the framework throws an ObjectNotFoundException exception. If the object is found, and a subitem is specified but not found, the proxy must throw a SubItemNotFoundException exception. The SubItemNotFoundException is detected by the framework, and it tries again automatically.

- Manage coordinate-based clicks

  If the input coordinates click an object or subitem that is beyond the region, change the coordinates to include the region in the object or subitem. Add a mechanism to generate a warning in the log when this occurs. The proxy might have to use coordinate-based clicks because the object screen layout cannot be completely described. For example, on a JTree, the PLUS_MINUS geometry is not known, but it can be found relative to the subitem to which it applies. It is acceptable for the proxy to expand the area of the subitem to include the PLUS_MINUS. You can use negative coordinates for subitems. Document all violations of the normal behavior. Ensure that no subitems with a specified coordinate-click click beyond the object.

- Managing exceptions when wrong objects are clicked

  If a wrong object is clicked, throw a com.rational.test.ft.CoordinateOnWrongObjectException exception. The wrong object might overlap the correct target. You can change the coordinates to avoid the child object within a container. Ignore the specified point and look for another point an appropriate object to click. Avoid recording coordinates on objects that have mappable children and do not allow clicks on the wrong objects.

- Managing exceptions when wrong subitems are clicked

  If a wrong subitem is clicked, throw a com.rational.test.ft.CoordinateOnWrongSubitemException exception. This exception does not apply to clicks associated with an object where a subitem is not specified. This type of click applies only when the object and any subitem in that object is clicked. In this case, the wrong subitem overlaps the correct target, and is probably a child. Avoid recording the coordinates on subitems that have nested subitems to avoid this exception. Ignore the specified point and look for another point on an appropriate subitem to click.

Customizing a script template

Rational® Functional TesterHCL OneTest™ UI uses its default script templates to format and provide basic information when you create a script. You can customize the information and format by customizing the script templates.

About this task

To customize a script template, use the Rational® Functional TesterHCL OneTest™ UI Script Templates Property Page. In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, you can also use the Java™ editor, which provides simple formatting of the template and help with the Java™ syntax. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, you can use the Code editor.

1. Right-click a functional test project in the Projects view, click **Properties > Rational Functional Tester > HCL OneTest UI** from the pop-up menu. Right-click a functional test project in the Solution Explorer, click **Properties > Rational Functional Tester > HCL OneTest UI** from the pop-up menu.
   The Rational® Functional TesterHCL OneTest™ UI Script Templates Property page is displayed.
2. In the **Select template type** list, click the script template to customize.
3. Edit the script template from the **Functional Test Script Templates** Property page:
   a. Edit the script template using the appropriate placeholder. For information about placeholders, see *Customizing Script Templates* related topic.
   b. Optionally, click **Apply** to save your edits as you work or when you finish changes to one script template.
   c. Click **OK** to save all edits to all script templates.

   You can also edit a script template with the Java™ editor in Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration or the Code Editor in Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration. Click **Open current template in Editor** to open the template in the appropriate editor.



Tip: To omit associated punctuation for any null properties, enclose the placeholder in a pair of carets (^). For example:

```
^%map:contextComment%^
```

Customizing script templates

You can customize the default script templates that Rational® Functional TesterHCL OneTest™ UI uses to format and provide basic information when you create a script.

To customize a script template, use the Rational® Functional TesterHCL OneTest™ UI. Script Templates Property Page. In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, you can also use the Java™ editor, which provides simple formatting of the template and help with the Java™ syntax. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, you can use the Code editor.

Several types of script templates are available:

Script: Header of the file -- Customizes the layout of new script files.

Script: Comment for Test Object -- Customizes a test object comment line inserted into a script by the recorder.

Script: Comment for top level Test Object -- Customizes a top-level test object comment line inserted into a script by the recorder.

Script: HTML Test Object Name -- Customizes the names of HTML test objects in a script.

Script: Java™ Test Object Name -- Customizes the names of Java™ test objects in a script.

Script: .Net Test Object Name -- In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, customizes the names of .NET test objects in a script.

Script: Windows® Test Object Name -- In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, customizes the names of Windows® test objects in a script.

VP: Verification Point Default Name -- Customizes the names of verification points that Rational® Functional TesterHCL OneTest™ UI generates by default in the Verification Point and Action Wizard.

Script Helper: Header of the file -- Customizes the layout of a helper class when auto-generated.

Script Helper: Test Object Method -- Customizes the layout of test object methods in the helper class.

Script Helper: Verification Point Method -- Customizes the layout of verification point methods in the helper class.

Script Helper Superclass -- -- Customizes the layout of the script helper superclass.

Script Helper Base Class -- Customizes the layout of the script helper base class.

You use placeholders in the script template to include information in each script that you generate. There are five types of placeholders:

- Default placeholders -- Available for any of the script templates.
- Script placeholders -- Use in the following types of script templates:
    - Script: Header of the file
    - Script Helper: Header of the file
    - Script Helper: Test Object Method
    - Script Helper: Verification Point Method
- VP placeholders-- Use in the VP: Verification Point Default Name script template.
- Object Map Property placeholders -- Use in the following types of script templates:
    - Script: Comment for Test Object
    - Script: Comment for top level Test Object
    - Script: HTML Test Object Name
    - Script: Java™ Test Object Name
    - Script: .Net Test Object Name
    - Script: Windows® Test Object Name
- Script Helper Super Class placeholders-- Use in the Script Helper Parent: Header of the file template.

The following table lists the placeholders available for each script template:

Script Template	Placeholder
Script: Header of the file	Default Placeholders and Script Placeholders
Script: Comment for Test Object	Default Placeholders and Object Map Property Placeholders
Script: Comment for top level Test Object	Default Placeholders and Object Map Property Placeholders

Script Template	Placeholder
Script: HTML Test Object Name	Default Placeholders and Object Map Property Placeholders
Script: .Net Test Object Name (Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration only)	Default Placeholders and Object Map Property Placeholders
Script: .Net and Windows® Test Object Name (Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration only)	Default Placeholders and Object Map Property Placeholders
Script: Java™ Test Object Name	Default Placeholders and Object Map Property Placeholders
VP: Verification Point Default Name	Default Placeholders, VP Placeholders, and Test Object Placeholders
Script Helper: Header of the file	Default Placeholders and Script Placeholders
Script Helper: Test Object Method	Default Placeholders and Script Placeholders
Script Helper: Verification Point Method	Default Placeholders and Script Placeholders
Script Helper Parent: Header of the File (Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration only)	Default Placeholders and Script Helper Super Class Placeholders

To omit associated punctuation for any null properties, enclose the placeholder in pairs of carets (^). Rational® Functional TesterHCL OneTest™ UI removes everything between the carets if the placeholder does not resolve to a valid value. For example, the underscore separator is removed if the #name property is null:

^%map:#name%_^%map:#role%

Default placeholders

You can use default placeholders to customize the systemwide placeholders such as properties, date, and time of any script template.

system placeholder

Usage: Use to customize any script template.

Syntax: %system:*argument*%

Description: Uses a Java™ system property as the argument.

Arguments for the **system** placeholder:

java.version -- Returns the Java™ Runtime Environment version.

java.vendor -- Returns the Java™ Runtime Environment vendor.

java.vendor.url -- Returns the Java™ vendor URL.

java.home -- Returns the Java™ installation directory.

java.vm.specification.version -- Returns the Java™ Virtual Machine specification version.

java.vm.specification.vendor -- Returns the Java™ Virtual Machine specification vendor.

java.vm.specification.name -- Returns the Java™ Virtual Machine specification name.

java.vm.version -- Returns the Java™ Virtual Machine implementation version.

java.vm.vendor -- Returns the Java™ Virtual Machine implementation vendor.

java.vm.name -- Returns the Java™ Virtual Machine implementation name.

java.specification.version -- Returns the Java™ Runtime Environment specification version.

java.specification.vendor -- Returns the Java™ Runtime Environment specification vendor.

java.specification.name -- Returns the Java™ Runtime Environment specification name.

java.class.version -- Returns the Java™ class format version number.

java.class.path -- Returns the Java™ class path.

java.ext.dirs -- Returns the Path of extension directory or directories.

os.name -- Returns the operating system name.

os.arch -- Returns the operating system architecture.

os.version -- Returns the operating system version.

file.separator -- Returns the File separator ("/" on UNIX®), ("\\" on Windows®).

path.separator -- Returns the Path separator (":" on UNIX®), (";" on Windows®).

line.separator -- Returns the Line separator ("\n" on UNIX®), ("\r/n" on Windows®).

user.name -- Returns the User's account name.

user.home -- Returns the User's home directory.

user.dir -- Returns the User's current working directory.

Examples:

%system:java.version% -- Returns the Java™ Runtime Environment version.

%system:java.vendor% -- Returns the Java™ Runtime Environment vendor.

%system:java.vendor.url% -- Returns the Java™ vendor URL.

date placeholder

Description: Returns the current date in the format specified and uses the following arguments with appropriate separators.

Syntax: %date: *argument*%

Comment: This placeholder follows the *java.text.SimpleDateFormat* format specification.

Arguments for the **date** placeholder:

yyyy -- Returns the year (4 digits).

yy -- Returns the year (2 digits).

MMM -- Returns the month (short name).

MM -- Returns the month (2 digits).

M -- Returns the month (1 or 2 digits).

EEEE -- Returns the day of week.

EE -- Returns the day of week short form.

dd -- Returns the day of month (2 digits).

d -- Returns the day of month (1 or 2 digits).

hh -- Returns the hour 1-12 (2 digits).

HH -- Returns the hour 0-23 (2 digits).

H -- Returns the hour 0-23 (1 or 2 digits).

KK -- Returns the hour 0-11 (2 digits).

K -- Returns the hour 0-11 (1 or 2 digits).

kk -- Returns the hour 1-24 (2 digits).

k -- Returns the hour 1-24 (1 or 2 digits).

mm -- Returns the minute.

ss -- Returns the second.

SSS -- Returns the millisecond.

a -- Returns the AM or PM.

zzzz -- Returns the time zone.

zz -- Returns the time zone (short form).

F -- Returns the day of week in month (that is, 3rd Thursday).

DDD -- Returns the day in year (3 digits).

D -- Returns the day in year (1, 2, or 3 digits).

ww -- Returns the week in year.

G -- Returns the era (BC or AD).

' -- Allows text within single quotes to appear in a script rather than interpreting the text as a command. (For example: 'dog' prevents dog from being processed).

'' -- Allows a single-quote character to appear in a script rather than interpreting the character as a command.

Examples:

Example of data placeholder	Result
%date:yyyy.MM.dd G 'at' hh:mm:ss z%	2005.07.10 AD at 15:08:56 PDT
%date:EEE, MMM d, ' ' yy%	Wed, July 10, '05
%date:h:mm a%	12:08 PM
%date:hh 'o''clock' a, zzzz%	12 o'clock PM, Pacific Daylight Time
%date:K:mm a, zz%	3:26 PM, PST
%date:yyyy.MMMMM.dd GGG hh:mm aaa%	2005.July.10 AD 12:08 PM

time placeholder

Description: Returns the current date in the format that you specify and uses the following arguments with appropriate separators. This placeholder extends the **date** placeholder. You can use the same **date** placeholder arguments, with the additional argument for milliseconds since 1970/01/01 00:00:00.000 GMT.

Syntax: %time: *argument*%

Comment: This placeholder follows the *java.text.SimpleDateFormat* format specification.

Additional argument for the **time** placeholder:

SSSS -- Returns milliseconds since 1970/01/01 00:00:00.000 GMT.

Example: %time:SSSS% -- Returns milliseconds since 1970/01/01.

env placeholder

Description: Uses an environment variable specified as the argument. Any number of environment variables are available on a system. These values are system dependent.

Syntax: %env: *argument*%

Arguments for the **env** placeholder:

PATH -- Returns the executables path.

TMPDIR -- Returns the temporary directory.

HOME -- Returns the users home directory.

Example: %env:PATH%

option placeholder

Description: Returns the value of a specified Rational® Functional TesterHCL OneTest™ UI customizable option for script execution.

Syntax: %option: *argument*%

 **Note:** For information about the available option arguments, see Modifying Options for Script Execution topic.

Examples:

%option:rt.project% -- Replaced by the Test Manager project name.

%option:rt.time.delay_before_gui_action% -- Replaced by the time delay before any user interface action is performed.

%option:rt.time.delay_before_mouse_down% -- Replaced by the time delay and inserted before a mouse down event is delivered.

static placeholder

Description: Invokes the specified parameterless static method. The static methods depend on the classes available in the user's Java™ environment.

Syntax: %static: *method*% where *method* is any visible complete Java™ static method specification.

An argument for the **static** placeholder includes:

java.lang.System.currentTimeMillis -- Returns the time since 1/1/1970.

Examples:

%static:java.lang.System.currentTimeMillis% -- Returns the time in milliseconds since 1/1/1970.

%static:com.rational.test.ft.script.ScriptUtilities.getOperatingSystemVersion -- Returns host-specific operating system version information.

Object map property placeholders

The object map property placeholders resolve object map placeholder values into property values. They also resolve default placeholder values.

Usage: Use the following placeholders to customize the following script templates:

- Script: Comment for Test Object
- Script: Comment for top level Test Object
- Script HTML Test Object Name
- Script Java™ Test Object Name
- Script: .Net Test Object Name
- Script: Windows® Test Object Name

map placeholder

Description: Resolves values relative to an entry in the object map. The map placeholder is only valid during helper script method generation and during recording to insert comments into the script, otherwise the test object instance is not known.

Syntax: %map: *property*%

Functional properties for the **map** placeholder:

context -- Returns the descriptive name of the closest parent registered in the Object Library as having context.

contextComment -- Returns the resolved context comment registered in the Object Library for the closest parent with context.

topContext -- Returns the descriptive name of the topmost parent unless this object does not have a parent.

topContextComment -- Returns the resolved context comment registered in the Object Library for the topmost parent.

Examples:

%map:context% -- Returns the descriptive name of the closest parent registered in the Object Library as having context.

The recognition property for the **map** placeholder returns the property name.

Syntax: %map: *RecognitionProperty*%

- The recognition properties are unique to each type of test object. To get the name of a recognition property for a particular test object to use with the object map placeholder, see the properties on the **Recognition** tab of the object map.
- Administrative properties are prefixed with a # character to signify that the property is an administrative rather than a recognition property. For information about administrative versus recognition properties, see Property Sets on page 1838 in the Test Object Map topic on page 1838.

Administrative properties for the **map** placeholder:

.class -- Returns the Java™ class name, the HTML tag (with an HTML prefix), or the VB class name of the test object.

#name -- Returns the test objects descriptive name.

#role -- Returns the test object role.

#domain -- Returns the domain in which the test object is defined, that is, Java™, HTML, or .NET.

#testobject -- Returns the interface class name used to interact with the test object.

#proxy -- Returns the proxy class name.

#description -- Returns a user-specified description, defined in the object map editor.

Examples:

%map:.class% -- Returns the Java™ class name, the HTML tag (with an HTML prefix), or the VB class name of the test object.

%map:#domain% -- Returns the domain in which the test object is defined, that is, Java™, HTML, or VB.

Script placeholders

Script placeholders resolve script-level placeholder values into script values. As the values are resolved, several lines of information can be cached, depending on the placeholder.

Usage: Use the following placeholders to customize the following script templates: Script: Header of the file, NameScript Helper: Header of the file, Script Helper: Test Object Method, Script Helper: Verification Point Method.

Comment: All script placeholder arguments are case insensitive.

script placeholder

Description: Resolves script placeholder values into script values.

Syntax: %script: *argument*%

Arguments for the script placeholder:

name -- Returns the name of the script (without a file suffix or package specification).

fullName -- Returns the full name of the script with package specification.

insertBefore -- Indicates the script code insertion point to be used by the recorder when creating a new script.

package -- Returns the name of the package containing the script.

packageDeclaration -- Returns the source for the package declaration, returns an empty string, " ", if the script is not in a package.

helper placeholder

Description: Resolves helper placeholder values into helper values.

Syntax: %helper:argument%

Arguments for the **helper** placeholder:

name -- Returns the name of the helper script.

fullName -- Returns the full name of the helper script including package specification.

insertBefore -- Indicates the helper test object methods insertion point to be used when generating the script helper.

package -- Returns the package declaration for the helper script.

packageDeclaration -- Returns the source for the helper package declaration, null if the helper is not in a package.

extends -- Returns a library configurable script base class.

methodName -- Returns the name of a helper method being inserted into a helper class.

testObjectInterfaceName -- Returns the test object class for a helper method being inserted into a helper class.

vpName -- Returns the name of a verification point method being inserted into a helper class.

testobject and map placeholders

Description: Both placeholders resolve values relative to an entry in the object map and are valid only during helper script method generation. The property for the **testobject** placeholder returns the property name.

Syntax: %testobject: *property*% or %map: *property*%

One recognition property is:

.class -- Returns the Java™ class name, the HTML tag (with an HTML prefix), or the VB class name of the test object.

- The recognition properties are unique to each type of test object. To get the name of a recognition property for a particular test object to use with the object map placeholder, see the properties on the **Recognition** tab of the object map.
- Administrative properties are prefixed with a # character. For information about administrative versus recognition properties, see Property Sets on page 1838 in the Test Object Map on page 1838 topic.

Administrative properties:

#name -- Returns the test object's descriptive name.

#role -- Returns the test object role.

#domain -- Returns the domain in which the test object is defined, that is, Java™, HTML, or .NET.

#testobject -- Returns the interface class name used to interact with the test object.

#proxy -- Returns the proxy class name.

#description -- Returns a user-specified description, defined in the object map editor.

Examples:

%testobject:.class% -- Returns the Java™ class name, the HTML tag (with an HTML prefix), or the VB class name of the test object.

%testobject:#domain% -- Returns the domain in which the test object is defined, that is, Java™, HTML, or .NET.

VP placeholders

VP placeholders resolve Test Object and verification point (VP) placeholder values into property strings. Default placeholders values are also resolved.

Usage: Use the following placeholder to customize the VP: Verification Point Default Name template:

vp placeholder

Description: Allows access to verification point attributes.

Syntax: %vp: *argument*%

Arguments for the vp placeholder:

type -- Returns the type of verification point being created. For example, **text** may be used for the Visible Text test of a Data verification point (Text).

description -- Returns the description of verification point as presented in the record UI. For example, **Visible Text** may be used for a Data verification point (Text).

testobject placeholders

Description: This placeholder resolves values relative to an entry in the object map. It is valid only during helper script method generation. The property for the **testobject** placeholder returns the property name.

Syntax: %testobject: *property*%

One recognition property:

.class -- Returns the Java™ class name, the HTML tag (with an HTML prefix), or the VB class name of the test object.

Note:

- Administrative properties are prefixed with a # character. For information about administrative and recognition properties, see Property Sets on page 1838 in the Test Object Map on page 1838 topic.
- From the VP: Verification Point Default Name script template, you can also access any dynamically available property, which is the actual test object in the software under test, not the mapped test object.

Administrative properties:

#name -- Returns the test object's descriptive name.

#role -- Returns the test object role.

#domain -- Returns the domain in which the test object is defined, that is, Java™, HTML, or .NET.

#testobject -- Returns the interface class name used to interact with the test object.

#class -- Returns the full class name of the test object.

#className -- Returns the simple class name for the test object, that is, the full class name without the package information.

#description -- Returns a user-specified description, defined in the object map editor. If this property is null, Rational® Functional TesterHCL OneTest™ UI uses the simple #className property.

#property -- Returns any test object property converted to its toString value. You can find the properties available for a test object in the test object map property set.

Script helper superclass placeholders

You can use the script helper superclass placeholders to customize the Script Helper Parent, the Header of the file template.

helpersuperclass placeholder

Description: Allows access to the superclass for Functional Test Script Helper.

Syntax: %helpersuperclass: *argument*%

Arguments for the script helper superclass placeholder:

packageDeclaration -- Returns the specification of the name space for the class.

name -- Returns the simple name of the class without name space specification.

Using the API to edit functional test scripts

These topics describe how you can take advantage of the Rational® Functional TesterHCL OneTest™ UI application programming interface (API) or the scripting framework to make changes to functional test scripts. As a general rule, the modifications that you make with the scripting SDK, should emulate as closely as possible, the user interactions to test with the application-under-test.

You can start by using the scripting framework to make some of these simple modifications to recorded functional test scripts:

- Change a user action, such as `object().drag() to object().click()`.
- Delete recorded commands.
- Place an often repeated sequence of actions into a method.

Writing messages to the log

A log is a file that contains the record of events that occur while a Functional Test script is playing back. There are several different methods you can use to write messages to the log.

Rational® Functional TesterHCL OneTest™ UI supports several types of log files, or no logging at all. You select the type of log file (HTML log, or text log) through the user interface. Each logged event has an associated message.

Rational® Functional TesterHCL OneTest™ UI automatically logs these events:

- Script start
- Script end
- Calls to the callScript method
- Calls to the startApplicaction method
- Timer start
- Timer end

- Exceptions
- Verification points

To use the scripting SDK to include your own general messages in whatever type of log you specified through the user interface, use the `logInfo` method, as shown in this example:

```
if(AnAWTButtonButton(p1,0)isEnabled())
{
    logInfo("AWT button is enabled.");
}
else
{
    logInfo("AWT button is not enabled.");
}
```

With the scripting framework, you can log a test result by using the `logTestResult` method. The first parameter is a headline that describes the test. The second parameter is the result of the test (`true` for pass, `false` for a failure),. An optional third parameter is for additional information. For example:

```
logTestResult("Text buffer comparison",
     TextField_text.equals(msExpect));
```

Here is another example that uses the third parameter for additional information:

```
if(TextField_text.equals(msExpect))
{
    logTestResult("Text buffer comparison", true);
}
else
{
    logTestResult("Text buffer comparison", false,
       "Expected '"+TextField_text+"' but found '"+msExpect+"'");
}
```

If you want to use the scripting framework to write an error message to the log, use the `logError` method:

```
catch (Exception e)
   {logError("Exception e = "+e.toString());}
```

With the scripting SDK, you can add a warning message to the log using the `logWarning` method:

```
logWarning("Your warning message goes here.");
```

Modifying options for script execution

Some scripting framework options that affect script execution can be specified through the user interface. Values that you set in the user interface persist as the default values from script to script. However, you can also use the scripting framework to set some of these options directly in the functional test script, for example, the amount of time between keystrokes.

A programmatically set value only lasts until the end of playback. After playback ends, the option reverts to the default value. Constants for these options are defined in the `com.rational.test.ft.script.IOptionName` interface. See the *Rational® Functional TesterHCL OneTest™ UI API Reference* for information about the `com.rational.test.ft.script.IOptionName` interface.

To retrieve the current value of an option, use the `getOption` method as follows:

```
Object x = getOption(IOptionName.option);
```

You can test the value of x to determine whether you want to change the option value during playback. To do so, use the `setOption` method, which has the following general format:

```
setOption(IOptionName.option,value);
```

You must specify a value of a type that applies to the option. The Rational® Functional TesterHCL OneTest™ UI IDE has a Content Assist feature that can be helpful here. In the earlier example, after entering `IOptionName`, press Ctrl +Space, or click **Edit > Content Assist** from the menu. A list of all the options open. You can use the arrow keys to scroll through the list, or type the first few letters of the option name if you know it. When you press Enter, the currently selected option name is inserted into your script.

With the scripting framework, you can also reset the value of an option back to the default value by using the `resetOption` method. For example, to change the delay between keystrokes during playback for a short time, you can script a sequence like this:

```
setOption(IOptionName.DELAY_BEFORE_KEY_DOWN, 0.3);
InputWindow().inputKeys("abcdefg123");
resetOption(IOptionName.DELAY_BEFORE_KEY_DOWN);
InputWindow().inputKeys("999");
```

Starting a test script from within a script

Test scripts can contain methods that invoke other test scripts. You might want to take advantage of this functionality by creating a test script that serves as a command file for a suite of scripts.

You can use the `callScript` method as follows:

```
....
// import statements and comments
import myscripts;// Added so script can find test3.

public class RegressionSuite extends RegressionSuiteHelper
{
public void testMain (Object[] args)
{
    callScript("test1");
    callScript(new test2());
    callScript("myscripts.test3");
}
}
```

Do not call the `testMain` method from another test script (for example, `test1.testMain(...);`). Rational® Functional TesterHCL OneTest™ UI would then be unable to ensure that each test script is invoked properly and has the expected event handling support.

Querying values of object properties

Components in the application-under-test, such as dialog boxes, command buttons, and labels, have associated pieces of information called properties. Properties have a name and a value. This topic provides examples of why you may want to modify your script to access an object property.

- You may want to compare previous versions of a value to the current value and to do so would require a calculation (such as factoring in a depreciation rate).
- Sometimes querying a property may return a reference to other objects. In cases like this, you might need to test the value of a property of the returned object. This kind of scenario cannot be handled through the user interface. See Unregistering references to test objects on page 1183 for more information.
- You also might want to branch in your Functional Test script based on the current value of a property.

You can retrieve the value of a property programmatically by calling the `getProperty` method, which has the following syntax:

```
Object getProperty(String propertyName);
```

The following example uses the `getProperty` method to test whether a value of a property is being captured and reproduced correctly. The call to `getProperty` retrieves the value of the text property associated with the `yourOrderHasBeenReceivedYourOr` object.

```


import resources.QueryingObjectHelper;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class QueryingObject extends QueryingObjectHelper
{
 /**
  * Script Name   : QueryingObject
  * Generated     : Jul 19, 2006 2:31:56 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/19
  * @author Administrator
```

```
  */
 public void testMain(Object[] args)
 {
  startApp("ClassicsJavaA");

  // Frame: ClassicsCD
  placeOrder().click();

  // Frame: Member Logon
  ok().click();

  // Frame: Place an Order
  cardNumberIncludeTheSpacesText().click(atPoint(28,6));
  placeAnOrder().inputChars("1234123412341234");
  expirationDateText().click(atPoint(9,2));
  placeAnOrder().inputChars("12/12");
  placeOrder2().click();

  //Waiting for Object
  yourOrderHasBeenReceivedYourOr().waitForExistence();

  //Querying the Object
  String confirmationText = (String)yourOrderHasBeenReceivedYourOr().getProperty("text");
  logTestResult(confirmationText, confirmationText.startsWith("Your order has"));

  yourOrderHasBeenReceivedYourOr().click();
  ok2().click();

  // Frame: ClassicsCD
  classicsJava(ANY,MAY_EXIT).close();
 }
}
```

Rational® Functional TesterHCL OneTest™ UI also supports a `setProperty` method, but do not use it unless you are sure of the result. This method calls internal methods that may violate the integrity of the application-under-test.

Unregistering references to test objects

Helper script methods refer to an object in the application-under-test by using the test object map. Rational® Functional TesterHCL OneTest™ UI finds such mapped objects each time a method is called on the object. In some cases; however, you might not want Rational® Functional TesterHCL OneTest™ UI to do this.

For instance, you might want to call many methods directly on the same object, and it would lose time for Rational® Functional TesterHCL OneTest™ UI to find the object each time a method was called on it. You can use the `TestObject.find` method to find an object without Rational® Functional TesterHCL OneTest™ UI calling any methods on the object. `TestObject.find` returns a new `TestObject` containing a different kind of reference to the object in the application-under-test. This reference is sometimes called a bound reference, a found reference, or a non-mapped reference.

A bound reference retains access to the object in the application-under-test until you explicitly unregister the reference. Rational® Functional TesterHCL OneTest™ UI unregisters bound references only when the entire playback

ends, not when the script ends. As long as a bound reference to the object exists, Rational® Functional TesterHCL OneTest™ UI may prevent the object in the application from being entirely free. For example, while you hold a bound reference to a Java™ object, the Java™ object is not garbage collected. You must explicitly unregister any bound references you create as soon as you do not need them any more.

In a normal Functional Test script, the only `TestObjects` containing mapped references are the methods from the helper scripts. All other `TestObjects` contain bound references and must be unregistered. For example, the method `TestObject.getTopParent` is explicitly declared to return a `TestObject`. Other methods are declared to return a `java.lang.Object`, but can return a `TestObject` that must be unregistered -- for example, `TestObject.getProperty`.

RationalTestScript contains several methods that remove references to `TestObjects`, including `com.rational.test.ft.script.RationalTestScript.unregister` and `unregisterAll`. See the *Rational® Functional TesterHCL OneTest™ UI API Reference* for information on these methods.

Objects that are returned from the application-under-test that are not `TestObjects` are objects that represent a value. The type of such an object is referred to as a value class. A value class is a copy of the object in the application-under-test, not a reference to an object in the application-under-test. Common examples of value classes are `java.lang.Integer` and `java.awt.Rectangle`.

The Rational® Functional TesterHCL OneTest™ UI recorder and wizards only generate code that returns value classes. For example, a property that you see in the Object Properties test case is a property whose value is a value class. You can call `TestObject.getNonValueProperties` to find the reference properties available for a particular object. You can call `TestObject.getMethods` to see the list of all the methods that you could invoke by calling `TestObject.invoke`.

Use caution when dealing directly with `TestObjects` that contain references to objects in the application-under-test, because doing so may create instability in the application. Unregister these `TestObjects` as soon as possible.

Handling ambiguous recognition

In some situations during playback, Rational® Functional TesterHCL OneTest™ UI might not be able to differentiate between two similar objects in the software that is being tested. This topic describes handling these situations.

For example, in HTML applications when more than one instance of a browser is active, differentiating one browser from another based on toolbar actions is impossible if the actions are recorded as they are in these examples:

```
BrowserToolbar_Back().click()
BrowserToolbar_Forward().click()
```

```
 BrowserToolbar_Back().Click();
 BrowserToolbar_Forward().Click();
```

In cases like this, Rational® Functional TesterHCL OneTest™ UI avoids ambiguous recognition by locating the toolbar button in the browser that is identified by its currently loaded document which is referred to as an anchor for the target object. For example:

```
BrowserToolbar_Back(Browser_htmlBrowser(Document_MyHomePage(),
  DEFAULT), DEFAULT).click();
```

The back button on the toolbar is anchored by the browser, which is anchored by the document named My HomePage. However, this example would not work if each instance of the browser has the same loaded document. Note that the helper script methods that take an anchor as an argument also require another argument that specifies the component's state (the DEFAULT argument in the example above). The default state for HTML objects is LOADED. For HTML components, the states LOADING and UNINITIATED are also possible. The default state for Java™ objects is SHOWING and ENABLED. Other supported state flags for Java™ objects are NOT_SHOWING and DISABLED.

In addition, you can identify the browser instance by using a `TestObject` reference for it, invoking the find method on the browser as follows (remember to unregister the test object when you are done).:

```
TestObject browserOne = Browser_htmlBrowser(Document_MyHomePage(),
  DEFAULT).find();
```

```
 Dim browserOne As TestObject = Browser_htmlBrowser(Document_MyHomePage(), DEFAULT_FLAGS).Find()
```

The browser toolbar commands in the test script would look like this example:

```
BrowserToolbar_Back(myBrowser, DEFAULT).click();
```

Another situation where ambiguous recognition can be an issue is when a test has more than one application running at the same time. During playback, commands such as `b5().click()` are ambiguous. Because the `startApp` command returns a process test object, this reference can be used to specify which application a particular command applies to. For example:

```
ProcessTestObject p1 = startApp("SwingTest");
ProcessTestObject p2 = startApp("TryIt");
...
//b5().click(); ambiguous on playback; which application?

b5(p1, DEFAULT).click();
```

```
 Dim p1 As ProcessTestObject = StartApp("SwingTest")
 Dim p2 As ProcessTestObject = StartApp("TryIt")

 'B5().Click(); ambiguous on playback; which application?

 B5(p1, DEFAULT_FLAGS).Click()
```

In the last line of the example, the process test object functions as an anchor to locate the desired application. Note that calling the unregister method for a process test object is unnecessary.

Ambiguous recognition can also occur in scenarios where two instances of the same control exist with identical sets of recognition properties. In such a case, the AmbiguousRecognitionException exception is thrown during playback. To handle the exception and to resolve which control needs to be clicked, you can use this code:

```
public class AmbiguousRecognitionTest extends AmbiguousRecognitionTestHelper
{
@Override
```

```
public void onAmbiguousRecognition(
ITestObjectMethodState testObjectMethodState, TestObject[] choices,
int[] scores) {
// TODO Auto-generated method stub
testObjectMethodState.setFoundTestObject(choices[0]);
// super.onAmbiguousRecognition(testObjectMethodState, choices, scores);
}
public void testMain(Object[] args)
{

aButton.click() // There are two aButton test objects visible on the screen, click the second button.

}
}
```

```
 Public Class AmbiguousRecognitionTest
  Inherits AmbiguousRecognitionTestHelper

  Public Sub onAmbiguousRecognition(ByVal testObjectMethodState As ITestObjectMethodState, ByVal choices
 As TestObject(), ByVal scores As Integer())

   testObjectMethodState.setFoundTestObject(choices(0))

  End Sub

  Public Function TestMain(ByVal args() As Object) As Object
   AButton().click()
   ' There are two aButton test objects visible on
   ' the screen, click the second button.
   Return (Nothing)
  End Function

 End Class
```

In this example, the second instance of the control is clicked.

Adding manual and dynamic verification points

In addition to verification points specified during recording, you can also incorporate new verification points into a Functional Test script. Scripting manual and dynamic verification points enables you to specify data for comparison against an object that is not found in the test object map. The data, however, must be value-class based.

For both the `vpManual` method and the `vpDynamic` method you can refer to the entry for `IFtVerificationPoint` in the *Rational® Functional TesterHCL OneTest™ UI API Reference* for information about restrictions on verification point names and data formats.

Manual verification points

Manual verification points are useful when you create the data for the verification point yourself, and you want to compare the data. For example, the data could be the result of a calculation or could come from an external source, such as a database.

Manual verification point objects are constructed using the `vpManual` method. When you call this method, you provide the data before `performTest` is executed. (The `performTest` method saves the supplied data, compares it when there is a baseline, and writes the result to the log.) The `vpManual` method has two signatures:

```
IFtVerificationPoint vpManual (java.lang.String vpName, java.lang.Object
actual)


IFtVerificationPoint vpManual (java.lang.String vpName, java.lang.Object
expected, java.lang.Object actual)
```

The first form of `vpManual` takes the name of the verification point and the actual data that is either compared to an existing baseline, or used to create a baseline if one does not already exist. Note that this value can be `null`. The *`vpName`* must be unique relative to the script. For example:

```
vpManual ("manual1", "The rain in Spain").performTest();
```

The second form of this method adds a parameter for the expected data to be compared with the actual. Either expected or actual can be null valued. For example:

```
vpManual ("manual1", "The rain in Spain", "The Rain in Spain").performTest();
```

In this example, the data does not match. The `performTest` method would record a verification point failure message in the log.

Dynamic verification points

Dynamic verification points are most useful when the `TestObject` is not mapped and not something that Rational® Functional TesterHCL OneTest™ UI would normally test, for example, an object that is not part of the application-under-test.

The `vpDynamic` method constructs dynamic verification points. Dynamic verification points raise the appropriate user interface the next time the script is played back. The user is able to insert verification point data tested against an object specified by the script. In this way, the user can avoid having to run the test manually to the appropriate state before recording the verification point. The `vpDynamic` method has two signatures:

```
IFtVerificationPoint vpDynamic (java.lang.String vpName)

IFtVerificationPoint vpDynamic (java.lang.String vpName, TestObject
objectUnderTest)
```

The first form of the `vpDynamic` method requires a unique (relative to the script) verification point name. The Recording Verification Point and Action wizard is raised the next time the script is played back. The user specifies the `TestObject` and the baseline data for subsequent runs to test against. The script must be run in interactive mode. For example:

```
vpDynamic("dynamic1").performTest();
```

The other form of the `vpDynamic` method requires specification of the `TestObject`. For example:

```
vpDynamic("dynamic1", AnAWTButtonButton()).performTest();
```

A modified UI, which does not display the `TestObject` hierarchy, appears on the first playback to specify data values for the baseline. While the specified `TestObject` does not have to be from the test object map, it must be consistently the same object for the results to be meaningful.

A common error when using these methods is to omit the `performTest` method. This is legal and compiles without warning, but no interesting action occurs when the script runs. For example:

```
vpDynamic("test1", AnAWTButtonButton()); //ERROR. Nothing happens.
```

Handling unexpected active Windows

A common problem in GUI testing is the appearance of an unexpected active window -- for example, a warning message box in an HTML browser. This topic describes how to handle this problem.

Imagine that you record a click action on a secure page, and this link takes you to a page that is not secure. Assume your browser's security setting is adjusted to cause a message box to appear, warning you that the next page will not be secure. You click **OK** to dismiss the warning message, and then you click a check box on the page that is not secure. The recorded Functional Test script would look something like this:

```
linkThatLeavesSecurePage().click();
Dialog_HtmlDialogButtonOK().click();
CheckboxOnTheUnsecurePage().click();
```

When you play the script back against a browser with a different security setting, the script does not play back because the `Dialog_HtmlDialogButtonOK()` cannot be found. You can comment out the `Dialog_HtmlDialogButtonOK().click();` statement, but you will have failures when the dialog *does* show up.

One solution is to wait for the message to appear. If it does not appear, you can continue. The solution can be achieved with the following code:

```
linkThatLeavesSecurePage().click();
try
{
    Dialog_HtmlDialogButtonOK().click();
}
catch(ObjectNotFoundException e) {}
CheckboxOnTheUnsecurePage().click();
```

This code accomplishes your primary goal. If the warning message appears, you dismiss it. If it does not appear, you eventually stop waiting and then continue. However, you may not want to wait the default amount of time for the warning message to show up. If you are sure that when the warning message does show up it will arrive within 5 seconds, you can speed this up by coding as follows:

```
linkThatLeavesSecurePage().click();
try
{
    Dialog_HtmlDialogButtonOK().waitForExistence(5,1);
    Dialog_HtmlDialogButtonOK().click();
}
catch(ObjectNotFoundException e) {}
CheckboxOnTheUnsecurePage().click();
```

A reasonable objection to this approach is that you need to add this special code wherever a link on a browser might switch pages and cause a change in security. Handling this situation in a common place without changing many test scripts would be more efficient. By implementing the `onObjectNotFound` exception you can handle the event whenever it occurs. By putting the implementation in a helper super script, you can handle the event for any Functional Test script that extends this helper super class.

The code in the following example implements a base class for scripts that test HTML applications. This base class implements `onObjectNotFound`. The `onObjectNotFound` method looks through all the HTML domains and looks for any HTML dialog boxes. Every HTML dialog box is dismissed by pressing Enter. If any dialog boxes are dismissed, the `TestObject` method is restarted. If no dialog boxes are dismissed, the method does nothing, and the `ObjectNotFoundException` is thrown as usual.

```
import com.rational.test.ft.script.*;
import com.rational.test.ft.object.interfaces.*;
/**
* This class provides some base capabilities for working
* with HTML.
*/
public abstract class HtmlScript extends RationalTestScript
{
/**
* Overrides the base implementation of onObjectNotFound. Whenever
* this event occurs, look through all the active domains (places
* where objects might be found). For HTML domains (Java
* and other domains are skipped) finds all the top objects.
* If the top object is an Html Dialog,
* types an Enter key to dismiss the dialog.
* Logs a warning when this happens.
*/
public void onObjectNotFound(ITestObjectMethodState testObjectMethodState)
{
   boolean dismissedAWindow = false;
   DomainTestObject domains[] = getDomains();
   for (int i = 0; i < domains.length; ++i)
   {
       if (domains[i].getName().equals("Html"))
       {
           // HTML domain is found.
           TestObject[] topObjects = domains[i].getTopObjects();
           if (topObjects != null)
           {
               try
               {
                   for (int j = 0; j < topObjects.length; ++j)
                   {
```

```
                        if (topObjects[j].getProperty(".class").equals("Html.Dialog"))
                        {
                            // A top-level HtmlDialog is found.
                            logWarning("HtmlScript.onObjectNotFound - dismissing dialog.");
                            try
                            {
                                dismissedAWindow = true;
                                ((TopLevelTestObject)topObjects[j]).inputKeys("{enter}");
                            }
                            catch(RuntimeException e) {}
                        }
                    }
                }
                finally
                {
                    //unregister all references to top objects
                    unregister(topObjects);
                }
            }

        }
    }
    if (dismissedAWindow)
    {
        //  try again
        testObjectMethodState.findObjectAgain();
    }
    else
    {
        logWarning("HtmlScript.onObjectNotFound; no Html Dialog to dismiss");
    }
}
}
```

Note that the above implementation of `HtmlScript` is only suitable for testing HTML. You may want to be able to use this base class for any script, including scripts testing Java™. In this case, you must make sure that the `TestObject` is a Rational® Functional TesterHCL OneTest™ UI `HTMLobject` before dismissing the HTML dialog boxes. You can add the following code to the beginning of the `onObjectNotFound` method:

```
if (!testObjectMethodState.getTestObject().
          getPropertyFromMap(IMapPropertyName.DOMAIN).equals("Html"))
{
    return;
}
```

Rational® Functional TesterHCL OneTest™ UI examples

Rational® Functional TesterHCL OneTest™ UI ships with some example code you can use in your own scripts. You can open these examples from within the sample project shipped with Rational® Functional TesterHCL OneTest™ UI.

To access the sample project, open the Samples Gallery from the Welcome Page, or from the **Help** menu. Click **Help > Samples Gallery**. In the gallery, browse to the **Rational Functional Tester Sample ProjectHCL OneTest UI Sample Project**, which is listed in the **Technology** category.

Rational® Functional TesterHCL OneTest™ UI sample project

The sample project was created using the Rational® Functional TesterHCL OneTest™ UI tutorial. You can look at the script and other test assets while you do the tutorial, or after you complete it, if you want to compare them to your files.

The sample project contains the script, verification points, object map, and other files that are created when you complete the tutorial. The sample script is in the state it would be in at the end of the tutorial. For reference, you can look at the script or other assets to compare them with your own, or actually play back the script. The tutorial script is called "ClassicsSample."

To open the project, click the **Import** link. The project will then be displayed in the Projects View. Click the script name to see it in the script window. You can play back the script by clicking the **Run Functional Test Script** button  on the product toolbar

In addition, the project has some example code you can use in your own scripts.

Examples

Class	Package	Description
ExtensionScript	superscript	Provides some general utility methods.
HtmlScript	superscript	Provides handler to automatically dismiss unexpected active HTML Dialogs.
WindowScript	superscript	Provides some methods that may be useful for getting around problems with native Microsoft® Windows® Applications.
SwtScript	superscript	Provides some methods that may be useful when testing SWT-based applications. Note that this implementation makes use of WindowScript, which is Microsoft-Windows specific. This class will not work on Linux®.
EclipseScript	superscript	Provides some methods that may be useful when testing plugins running inside the Eclipse platform (see http://www.eclipse.org/). Note that this code makes use of internal Eclipse classes, and consequently may break with future versions of eclipse. This class illustrates invoking static methods in the SUT and using custom Test Objects.
WorkbenchTestObject	testobject.eclipse	A Test Object for the Eclipse (see http://www.eclipse.org/) shell Workbench.
WorkbenchWindowTestObject	testobject.eclipse	A Test Object for the Eclipse (see http://www.eclipse.org/) shell WorkbenchWindow.
WorkbenchPageTestObject	testobject.eclipse	A Test Object for the Eclipse (see http://www.eclipse.org/) shell WorkbenchPage.

Using the examples

If you would like to use these examples, copy the testobject and superscript directories and their contents into a Functional Test project.

To use one of the superscripts, set the helper superclass property for a script to the full class name of the superscript. For example, to use the ExtensionScript superclass, for a script called X, right-click X in the Functional Test Projects View and select Properties from the popup menu. In the dialog titled "Properties for X.java", select "Functional Test Script" in the list on the left. Finally, set the text in the edit box labeled "Helper superclass" to "superscript.ExtensionScript". You can also modify your project preferences so that all newly created scripts in the project will extend this superscript. To define a default helper superscript for a project, right-click the project and select Properties from the popup menu, then set the text field in "Rational® Functional TesterHCL OneTest™ UI Project" labeled "New Script Helper superclass".

Once this is done, your X.java script can make use of the additional methods of ExtensionScript such as `getClipboardText()`, `setClipboardText()`, `clipboardVP()`, etc.

Determining the values of cells in a table

When working with Java™ or HTML tables, you might want to extract the value of a given cell in the table. There are many ways to do this; one simple approach is to query the table directly.

The example shows how to create custom Java™ code that exploits the Functional Test object model to extract the information from a table. The sample first uses the `getTestData` method to have Rational® Functional TesterHCL OneTest™ UI return a `TestDataTable` object that contains all of the data in the table. Given this data table, the `getRowCount` and `getColumnCount` methods determine the size of the table. Finally, with these numbers, the code cycles through each cell and uses the `getCell` method to determine the contents of each cell in the table. The values in the cells display in the console window.

```
import resources.TableTestHelper;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class TableTest extends TableTestHelper
{
 /**
  * Script Name   : TableTest
  * Generated     : Jul 17, 2006 1:56:28 PM
  * Description   : Functional Test Script
```

```
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/17
  * @author Administrator
  */
 public void testMain(Object[] args)
 {
  startApp("ClassicsJavaA");

  // Frame: ClassicsCD
  jmb().click(atPath("Order"));
  jmb().click(atPath("Order->View Existing Order Status..."));

  // Frame: View Order Status
  nameComboB().click();
  nameComboB().click(atText("Claire Stratus"));
  ok().click();

  // Frame: View Existing Orders
  existingTable().click(atCell(atRow("ORDER ID", "7", "ORDER DATE", "3/11/98", "STATUS", "Order
Initiated"), atColumn("ORDER ID")), atPoint(33,2));

  // Query object to find out what kind of data it has.
    System.out.println (existingTable().getTestDataTypes());

    //Declare variable for table.
    ITestDataTable myTable;
    myTable = (ITestDataTable)existingTable().getTestData("contents");

    //Print out total rows & columns.
    System.out.println ("Total Rows: " + myTable.getRowCount());
    System.out.println ("Total Cols: " + myTable.getColumnCount());

    //Print out cell values.
    for (int row =0;row < myTable.getRowCount();row++)
    {
        for (int col = 0;col < myTable.getColumnCount();col++)
        {
            System.out.println("Value at cell (" + row+ "," + col+")is: " + myTable.getCell(row,col));
        }
    }

  close().drag();

  // Frame: ClassicsCD
  classicsJava(ANY,MAY_EXIT).close();
 }
}
```

Reading the Windows registry

The Windows® registry is a database used by the Windows® operating system to store configuration information. Often it becomes necessary for a tester to read information out of this database using Rational® Functional TesterHCL OneTest™ UI commands. This topic provides examples for doing this.

The following example is applicable for scripts running on Windows®:

```
import javax.swing.JOptionPane;

import resources.RegistryExampleHelper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class RegistryExample extends RegistryExampleHelper
{
 /**
  * Script Name   : RegistryExample
  * Generated     : Jul 20, 2006 1:48:49 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/20
  * @author Administrator
  */
 public void testMain (Object[] args)
 {
     try
     {
         //Use this code to extract String (REG_SZ) values from the registry.
      String regKeyString ="HKEY_LOCAL_MACHINE\\SOFTWARE\\Rational Software\\Rational Test\\8\\Rational
 FT Install Directory";
              //Use this code to extract String (REG_SZ) values from the registry.
      String regKeyString ="HKEY_LOCAL_MACHINE\\SOFTWARE\\HCL Technologies\\HCL OneTest UI Install
 Directory";

         String regValueString = getOperatingSystem().getRegistryValue(regKeyString);
         JOptionPane.showMessageDialog(null, regValueString,"String Registry Value",1);
     }
     catch (NoSuchRegistryKeyException e)
     {
         JOptionPane.showMessageDialog(null, "Error finding registry key.");
         System.out.println ("No Such Registry Key Exception." + e);
     }
     try
     {
         //Use this code to extract Integer (DWORD) values from the registry.
         String regKeyInt = "HKEY_CURRENT_USER\\Control " +"Panel\\Desktop\\LowLevelHooksTimeout";
         Integer regValueInt = new
            Integer(getOperatingSystem().getRegistryIntValue(regKeyInt));
         JOptionPane.showMessageDialog(null,regValueInt, "Integer Registry " + "Value ",1);
     }
     catch (NoSuchRegistryKeyException e)
```

```
    {
        JOptionPane.showMessageDialog(null, "Error finding registry key.");
        System.out.println ("No Such Registry Key Exception. (" + e + ")" );
    }
}
}
```

There are two commands available to read values from the registry. The `getRegistryValue` command is used to read string values from the registry. The `getRegistryIntValue` is used to read integer values from the registry. The terms "REG_SZ" describe the string and integer types. Both of the commands take a type `String` argument, which contains the registry key to extract.



Note: When entering keys, the "\" is a special character in Java™ and must be doubled to "\\" to be taken as a literal.

The example extracts both a string and an integer value from the registry. Looking first at the `String` value segment, notice the core code:

```
String regKeyString ="HKEY_LOCAL_MACHINE\\SOFTWARE\\Rational Software\\Rational Test\\8\\Rational FT
 Install Directory";
String regValueString = getOperatingSystem().getRegistryValue(regKeyString);
JOptionPane.showMessageDialog(null, regValueString,"String Registry Value",1);
```

```
String regKeyString ="HKEY_LOCAL_MACHINE\\SOFTWARE\\HCL Technologies\\HCL OneTest UI\\HCL Install
 Directory";
String regValueString = getOperatingSystem().getRegistryValue(regKeyString);
JOptionPane.showMessageDialog(null, regValueString,"String Registry Value",1);
```

The first line creates a type `String` variable, which contains the registry value to extract. The second line executes the command and stores it in the type `String` variable `regValueString`. The third line uses the `JOptionPane.showMessageDialog` class to display the registry value in a message box on the screen. For those unfamiliar with this last class, it is a Java™ Swing class, which must be imported to be available. Note the last `import` statement at the top of the script.

The second segment extracts the type `int` value. In the example, the simple type `int` is converted to an `Integer` object, so that it can be displayed in the `JOptionPane` dialog. Otherwise, the code is identical to the first segment.

Both of the commands throw a `NoSuchRegistryKeyException` when they fail. Therefore, it is a good idea to wrap these methods within a try/catch block, as in the example. You can change the registry key to one that does not exist and run the script. You will see an error message indicating the key could not be found.

Iterating through items in a tree control using the getTestData method

This topic provides an example of using Rational® Functional TesterHCL OneTest™ UI's `getTestData` method to programmatically access the values on the branches of a tree control.

The following example tests against the Classics Java™ application:

```
import resources.GetTreeDataExampleHelper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class GetTreeDataExample extends GetTreeDataExampleHelper
{
 /**
  * Script Name   : GetTreeDataExample
  * Generated     : Jul 14, 2006 4:46:31 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/14
  * @author Administrator
  */
 public void testMain(Object[] args)
 {
  //Start Classics Java Application
  startApp("ClassicsJavaA");

  // Frame: ClassicsCD
  tree2().waitForExistence();

  //Display available test data types available from tree
  System.out.println ("Available Tree Data Types: " + tree2().getTestDataTypes());

  //Declare variables for tree
  ITestDataTree cdTree;
  ITestDataTreeNodes cdTreeNodes;
  ITestDataTreeNode[] cdTreeNode;

  //Variables to hold tree data
  cdTree = (ITestDataTree)tree2().getTestData("tree");
  cdTreeNodes = cdTree.getTreeNodes();
  cdTreeNode = cdTreeNodes.getRootNodes();

  //Print out total number of nodes
  System.out.println ("Tree Total Node Count: " + cdTreeNodes.getNodeCount());
  System.out.println ("Tree Root Node Count : " + cdTreeNodes.getRootNodeCount());

  //Iterate through tree branches; this is a recursive method.
  for (int i = 0;i<cdTreeNode.length;++i)
  showTree(cdTreeNode[i], 0);

  //Shut down Classics Java Application
  classicsJava(ANY,MAY_EXIT).close();
  }
```

```
  void showTree(ITestDataTreeNode node, int indent)
  {
  //Recursive method to print out tree nodes with proper indenting.

  //Determine number of tabs to use - to properly indent tree
  int tabCount = ( indent < tabs.length() ? indent :
  tabs.length() );

  //Print out node name + number of children
  System.out.println(tabs.substring(0, tabCount) + node.getNode() + " (" + node.getChildCount() +
 "children)" );

  //Determine if node has children; recursively call this same
  //method to print out child nodes.
  ITestDataTreeNode[] children = node.getChildren();
  int childCount = ( children != null ? children.length : 0 );
  for ( int i = 0; i < childCount; ++i )
  showTree(children[i], indent+1);
  }

  //String of tabs used to indent tree view
  final String tabs = "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t";

}
```

On the first screen of this application is a Java™ Swing `JTree` component, which lists five composers. The next level down lists CDs available for the selected composer. The code in this sample extracts the values from all of the branches of the tree and displays them in the console window.

The first step to extracting the data is to use the `getTestData` method to extract the data from the control. This is done with the following syntax:

```
ITestDataTree cdTree;
cdTree = (ITestDataTree)tree2().getTestData("tree");
```

The next step is to create an array that contains all of the nodes on the tree. This is done as follows:

```

ITestDataTreeNodes cdTreeNodes;
ITestDataTreeNode[] cdTreeNode;

cdTreeNodes = cdTree.getTreeNodes();//Encapsulates the root
nodes.
cdTreeNode = cdTreeNodes.getRootNodes();;//Extracts actual
root nodes.
```

Note that this is a two-step process. First, you must use the `getTreeNodes` method to return a TreeNodes object. Then you can call the `getRootNodes` method to extract an array of the root nodes for the tree.

With the tree nodes in hand, you can use recursion to walk though each node to determine its value and the number of direct children it contains. This is done in the recursive method `showTree`. A recursive method is a method that calls itself, and is an efficient way to walk through a tree structure. To extract the value of the node, the `getNode` method is

used. To extract the number of children contained by the node, the `getChildCount` method is used. In the example, this is done with the following code:

```
System.out.println(tabs.substring(0, tabCount) + node.getNode()+" (" + node.getChildCount() + "
 children)");
```

Note the additional coding provided in the custom `showTree` method is to enable a formatted printing using tabs to indicate the indentation of the tree.

Iterating through table cells using the getTestData method

This topic provides an example of using Rational® Functional TesterHCL OneTest™ UI's `getTestData` method to access the values in the cells of a grid control.

The example tests against the Classics Java™ application:

```
import resources.GetGridDataExampleHelper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
* Description   : Functional Test Script
* @author Administrator
*/
public class GetGridDataExample extends GetGridDataExampleHelper
{
/**
* Script Name   : GetGridDataExample
* Generated     : Jul 14, 2006 3:05:22 PM
* Description   : Functional Test Script
* Original Host : WinNT Version 5.1  Build 2600 (S)
*
* @since  2006/07/14
* @author Administrator
*/
public void testMain(Object[] args)
{
//Start Classics Java Application
startApp("ClassicsJavaA");

//Navigate to Existing Order Grid
jmb().click(atPath("Order"));
jmb().click(atPath("Order->View Existing Order Status..."));

// Frame: View Order Status
nameComboB().click();
nameComboB().click(atText("Claire Stratus"));
ok().click();
```

```
// Frame: View Existing Orders
existingTable().click(atPoint(172,92));

//Get the data for the table
ITestDataTable orderTable = (ITestDataTable)existingTable().getTestData("contents");

//Display the available data types for the grid, total rows and columns.
System.out.println ("Available Data Types: " + existingTable().getTestDataTypes());
System.out.println ("Total Rows in table : " + orderTable.getRowCount());
System.out.println ("Total Cols in table : " + orderTable.getColumnCount());

    // Cycle through all rows
    for (int row=0; row < orderTable.getRowCount();++row)
    {
        // Cycle through all columns
        for (int col=0; col < orderTable.getColumnCount();++col)
        {
            // Print out values of cells at (row,col) coordinates
            System.out.println ("Row " + row + ", " + orderTable.getColumnHeader(col) + ": "
 +orderTable.getCell(row,col) );
     }
   }
// Close the frame
close().click();

// Frame: ClassicsCD
classicsJava(ANY,MAY_EXIT).close();
}
}
```

This example navigates to the "View Existing Orders" screen of the application. The code in this sample extracts the values from all cells in the grid and displays them in the console window.

The first step to extracting the data is to use the `getTestData` method to extract the data from the control. This is done with the following syntax:

```
ITestDataTable orderTable;
orderTable = (ITestDataTable)existingTable().
  getTestData("contents");
```

Given this data set, you can determine the total number of rows and columns by using the `getRowCount` and `getColumnCount` methods. You can also ask the control what data types are available from the table using the `getTestDataTypes`. The following code sends the results of these queries to the console window.

```
System.out.println ("Available Data Types: " +
     existingTable().getTestDataTypes());
System.out.println ("Total Rows in table : " +
     orderTable.getRowCount());
System.out.println ("Total Cols in table : " +
     orderTable.getColumnCount());
```

The next step is to print out the values of the individual cells, which is done by using a `for` loop to cycle through the rows and columns of the grid:

```
for (int row=0; row < orderTable.getRowCount();++row)
{
    // Cycle through all columns
    for (int col=0; col < orderTable.getColumnCount();++col)
    {
        // Print out values of cells at (row,col) coords
        System.out.println ("Row " + row + ", " +
        orderTable.getColumnHeader(col) + ": " +
        orderTable.getCell(row,col) );
    }
}
```

The example script uses the `getCell` method to print out the value of the current cell. Note also that the `getColumnHeader` method prints out the current column header. When working with a grid, the numbering for both rows and columns starts at 0. This does not apply to the `getRowCount` and `getColumnCount` methods where numbering starts at 1.

Passing parameters to the callScript method

This topic describes how to use the different signatures of the `callScript` method to pass data from one script to another.

The example uses two different Functional Test scripts:

- TheCaller, which calls another script and passes parameters
- TheCalled, which receives the parameters and prints them to `System.out`

TheCaller script uses three different versions of the `callScript` method:

- **Without additional parameters:** This is the default usage of the `callScript` method, which will execute the specified script.

```
callScript("TheCalled");
```

- **With additional string array parameter:** An array of strings is used to pass string parameters to the called script.

```
String[] dataToPass = new String[4];
...
callScript("TheCalled",dataToPass);
```

- **With additional object array parameter:** An array of objects is used to pass different object type parameters to the called script.

```
Object[] objdataToPass = new Object[4];
...
callScript("TheCalled",objdataToPass);
```

The `TheCaller` script was recorded as follows:

```
import resources TheCallerHelper;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
* Description : Functional Test Script
* @author Administrator
*/

public class TheCaller extends TheCallerHelper
{
/**
  * Script Name   : TheCaller
  * Generated     : Jul 14, 2006 5:13:02 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/14
   * @author Administrator
*/
public void testMain (Object[] args)
{

    callScript("TheCalled");

    String[] dataToPass = new String[4];
    dataToPass[0] = "this";
    dataToPass[1] = "is";
    dataToPass[2] = "really";
    dataToPass[3] = "cool";

    callScript("TheCalled",dataToPass);

    Object[] objdataToPass = new Object[4];
    objdataToPass[0] = new String("Thought the previous was cool?");
    objdataToPass[1] = "Take this one!";
    objdataToPass[2] = new Float(0.02);
    objdataToPass[3] = new Integer(4711);

    callScript("TheCalled",objdataToPass);
}
}
```

The `TheCalled` script uses a simple loop to print the received parameters to `System.out`:

```
import resources.TheCalledHelper;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;
/**
* Description : Functional Test Script
* @author Administrator
*/

public class TheCalled extends TheCalledHelper
{
/**
  * Script Name   : TheCalled
  * Generated     : Jul 14, 2006 5:13:02 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/07/14
  * @author Administrator
*/
public void testMain (Object[] args)
{
    if (args.length < 1)
    {
        System.out.println( "Expected at least 1 arg, but I got:
             "+args.length);
        return;
    }
    else
    {
        System.out.println( "Got: "+args.length+" args");
    }

    for (int i = 0; i < args.length; ++i)
    {
        System.out.println( " arg["+i+"] = "+args[i]);
    }
}
}
```

Extracting data from a combobox/list control (JComboBox)

You can use Rational® Functional TesterHCL OneTest™ UI's `getTestData` method to access the values in the list of a ComboBox/List control.

The following example tests against the Classics Java™ application:

```
import resources.GetListDataExampleHelper;

import com.rational.test.ft.*;
```

```
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class GetListDataExample extends GetListDataExampleHelper
{
/**
* Script Name   : GetListDataExample
* Generated     : May 16, 2006 9:06:46 AM
* Description   : Functional Test Script
* Original Host : WinNT Version 5.1  Build 2600 (S)
*
* @since  2006/05/16
* @author Administrator
*/
public void testMain(Object[] args)
{
startApp("ClassicsJavaA");

// Frame: ClassicsCD
tree2().click(atPath("Composers->Schubert->Location(PLUS_MINUS)"));
tree2().click(atPath("Composers->Schubert->Die schone Mullerin, Op. 25"));
placeOrder().click();

//Declare variables for list
ITestDataList nameList;
ITestDataElementList nameListElements;
ITestDataElement nameListElement;

//Frame: Member Logon
nameCombo().waitForExistence();

//Available test data types: {selected=Selected List Element,
//list=List Elements}
java.util.Hashtable ht = nameCombo().getTestDataTypes();
System.out.println(ht);

//Get all elements
nameList = (ITestDataList)nameCombo().getTestData("list");
nameListElements = nameList.getElements();

int listElemCount = nameList.getElementCount();

 for (int i = 0; i < listElemCount; i++)
 {
 nameListElement = nameListElements.getElement(i);
 System.out.println(nameListElement.getElement());

 // Click on each element
 nameCombo().click();
```

```
 nameCombo().click(atText(nameListElement.getElement().toString()));
 };

cancel().click();

// Frame: ClassicsCD
classicsJava(ANY,MAY_EXIT).close();
}
}
```

This example first opens up the Classics Java™ application. It selects a composer in the tree and an album (composer = Schubert, album = "Die Schone Muellerin") and clicks the **Place Order** button. In the next screen (Member Login - dialog) the sample code extracts the list of values from the ComboBox and displays them in the console window before clicking on each list element.

The first step is to extract the data from the control by using the `getTestData` method:

```
ITestDataList nameList;
nameList = (ITestDataList)nameCombo().getTestData("list")
```

To find out which data types are available for a control, use the following code:

```
java.util.Hashtable ht = nameCombo().getTestDataTypes();
```

Given this data set, you can create an array that contains all of the elements of the list. This is done as follows:

```
ITestDataElementList nameListElements;
nameListElements = nameList.getElements();
```

With the list elements in hand, you can create a loop that accesses each list element. To determine the number of list elements, use the `getElementCount` method. To extract the value of the list element, the `getElement` method is used. In the example, this is done with the following code:

```
int listElemCount = nameList.getElementCount();

for (int i = 0; i < listElemCount; i++)
{
  nameListElement = nameListElements.getElement(i);
  System.out.println(nameListElement.getElement());

  // Click on each element
 nameCombo().click();
 nameCombo().click(atText(nameListElement.getElement().toString()));
};
```

Playing back low level mouse and keyboard actions

Low-level playback of mouse and keyboard actions provides more control of events of user actions. For example, Rational® Functional TesterHCL OneTest™ UI currently supports TestObject.click(), where the click consists of low-

level actions including mouse move, left mouse button down, and left mouse button up. With this functionality, you can play back the individual components of a mouse click.

Low-level playback also supports mouse wheel scrolling.

You may want to use low-level playback to overcome a product limitation or an obscure mouse or keyboard action. For example, to draw a circle on a canvas in a drawing program, Rational® Functional TesterHCL OneTest™ UI does not support the complex circular drag but you can use the drag() method to draw straight lines. To overcome an obscure mouse or keyboard action, you can use low-level playback to play back the mouse actions for drawing the circle.

The RootTestObject class includes two methods:

- emitLowLevelEvent(LowLevelEvent)
- emitLowEvent(LowLevelEvent[])

Factory methods on SubitemFactory for construction of LowLevelEvents include:

- delay(int)
- keyDown(string)
- keyUp(string)
- mouseMove(point)
- mouseWheel(int)
- leftMouseButtonDown()
- leftMouseButtonUp()

Parallel methods exist for the middle and right mouse buttons. The delay event guarantees a delay of at least the milliseconds specified, taking into account the time taken by the system to consume the previous event.

A Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration example to draw the letter V in the upper left portion of the drawing canvas:

```
// This routine will draw a "V" in the upper left portion
// of the drawing canvas.
// First a point in the upper left corner will be clicked, the left mouse
// button will be held down for the duration of the action, the mouse
// will be moved to the right and down, then to the right and back up,
// and finally the left mouse button will be released.
Rectangle screenRect =
   (Rectangle) drawingWindow().getProperty(".screenRectangle");
Point origin = new Point(screenRect.x + 5, screenRect.y + 5);
LowLevelEvent llEvents[] = new LowLevelEvent[7];
llEvents[0] = mouseMove(atPoint(origin.x, origin.y));
llEvents[1] = leftMouseButtonDown();
// insert a delay to provide the SUT with time to respond
// to the events being delivered to it.
llEvents[2] = delay(250);
llEvents[3] = mouseMove(atPoint(origin.x + 25, origin.y + 50));
llEvents[4] = delay(250);
```

```
llEvents[5] = mouseMove(atPoint(origin.x + 50, origin.y));
llEvents[6] = leftMouseButtonUp();
getRootTestObject().emitLowLevelEvent(llEvents);
```

A Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration example to test a TrackBar control and confirm the control responds to the mouse wheel events:

```
' This will test a TrackBar control to make sure
' that it responds to mouse wheel events.
TrackBar1Slider().Click(AtPoint(0, 0))
' Create a Low Level Event representing scrolling
' the mouse wheel down 25 clicks.
Dim ScrollDown As LowLevelEvent = MouseWheel(-25)
GetRootTestObject().EmitLowLevelEvent(ScrollDown)
' Verify The Results.
```

Searching for test objects

You can search for one or more test objects that match specified search criteria. The search is based on name/value pairs that represent properties of the test object or test objects you are looking for. The search can either be global, or limited to children of a parent test object.

A `RootTestObject` object represents a global view of the software being tested. To perform a global search, you invoke the find method on the `RootTestObject` object. Invoking a find method on a test object only searches the children of that test object.

The first argument in the find method is a subitem for the search properties. The second optional argument is a flag that indicates whether only children that might be included in the test object map should be searched. The following values for the property subitems are valid:

- `atProperty` -- A name/value pair representing a test object property
- `atChild` -- One or more properties that must be matched against the direct child of the starting test object
- `atDescendant` -- One or more properties that can be matched against any child of the starting test object
- `atList` -- A sequential list of properties to match against. These subitems for the `atList` value are valid:
    - `atChild`
    - `atDescendant`
    - `atProperty`

  The first list item is matched against to get a list of candidates, and out of those candidates, their descendants are matched against for the next list item, and so on.

Special properties apply to the RootTestObject.find method, including these properties:

- `.processName` : This top-level property has two functions:
    - Dynamically enable the processes with that process name
    - Constrain the find method to only look in processes with that name
- `.processId`: This top-level property has two functions:

- Dynamically enable the processes with that process ID (pid)
- Constrain the find to only look in processes with that process ID (pid)

Note: The `.processId` property is valid for dynamically enabled domains like Microsoft .NET and Windows. It is not valid for enabled domains, such as HTML and Java.

- `.domain`: This flag specifies to only search in top-level domains that match the domain property
- `.hWnd`: When the `hWnd` property is used for the search, if the "Win" `.domain` property is also specified, the matching window is enabled for testing with the Windows® domain.
- `Handle`: When the window handle property is used for the search, if the "Net" `.domain` property is also specified, the matching window is enabled for testing with the Microsoft .NET domain.

Examples:

```
TestObject[] foundTOs ;
RootTestObject root = RootTestObject.getRootTestObject() ;
// Find all top-level windows in the Windows domain that have the caption "My Document"
CaptionText caption = new CaptionText("My Document") ;
foundTOs = root.find(atChild(".domain", "Win", ".caption",
     caption)) ;

// Find any dialog boxes, and then return their children "OK" buttons.
RegularExpression dialogRE = new
     RegularExpression("*dialog", false) ;
RegularExpression buttonRE = new
     RegularExpression("*button", false) ;
foundTOs = root.find(atList(atDescendant(".class",
                     dialogRE),
                     atChild(".class", buttonRE,".value",
                     "OK"))) ;

// Start Notepad, dynamically enable that process, find its top-level window that matches the process ID
 and get its descendant text window.
 ProcessTestObject p1 = StartApp("Notepad") ;
 Integer pid = new Integer((int)p1.getProcessId()) ;
 foundTOs = root.find(atList(atProperty(".processId",
     pid), atDescendant(".class", ".text"))) ;

// This enables a Windows application with the provided window handle and returns a test object that
 represents the window.
Long hWnd = getAppsHwnd();
foundTOs = root.find(atChild(".hwnd", hWnd, ".domain", "Win"));

// This enables a .NET application  with the provided window handle and returns a test object that
 represents the window.
Long handle = getAppsHwnd();
foundTOs = root.find(atChild("Handle", handle, ".domain", "Net"));
```

Examples:

```
Dim FoundTOs As TestObject()
```

```
Dim Root As RootTestObject
Root = RootTestObject.GetRootTestObject()

' Find all top-level windows in the Windows domain that have the caption "My Document"
Dim Caption As CaptionText
Caption = New CaptionText("My Document")
FoundTOs = Root.Find(AtChild(".Domain", "Win", ".caption", Caption))

' Find any dialog boxes, and then return their children "OK" buttons.
Dim DialogRE As RegularExpression = New RegularExpression("*dialog", False)
Dim ButtonRE As RegularExpression = New RegularExpression("*button", False)
FoundTOs = Root.Find(AtList( AtDescendant(".class", DialogRE), AtChild(".class", ButtonRE, ".value",
 "OK"))


' Start Notepad, dynamically enable that process, find its top-level window that matches the process ID,
 and get its descendant text window.
Dim P1 As ProcessTestObject = StartApp("Notepad")
Dim Pid As Integer = P1.GetProcessId()
FoundTOs = Root.Find(AtList(AtProperty(".processId", Pid), AtDescendant(".class", ".text")))

' This enables a Windows application with the provided window handle and returns a test object that
 represents the window.
Dim HWnd As Long = GetAppsHwnd()
FoundTOs = Root.Find(AtChild(".hwnd", HWnd, ".domain", "Win"))

' This enables a .NET application with the provided window handle and returns a test object that
 represents the window.
Dim Handle As Long = GetAppsHwnd()
FoundTOs = Root.Find(AtChild("Handle", Handle, ".domain", "Net"))
```

Rational® Functional TesterHCL OneTest™ UI dynamically enables the Windows and .NET applications by using the `.processName` property. To find the required test object on a Windows or .NET application, use the `.processName` property in the query.

Example: The following example code finds the number 9 button in a calculator and then clicks it.

```
Property[] props = new Property[4];
       // Find the top-level window of calculator application
       props[0] = new Property(".processName", "calc.exe");
       props[1] = new Property(".class","SciCalc");
       props[2] = new Property(".name", "Calculator");
       props[3] = new Property(".text", "Calculator");
       TestObject[] tos = find(atChild(props));

       if(tos.length > 0)
       {
           // Find button that contains the text 9
           props = new Property[3];
           props[0] = new Property(".class","Button");
           props[1] = new Property(".name", "9");
           props[2] = new Property(".text", "9");
           TestObject[] tos9 = tos[0].find(atChild(props));

           if(tos9.length > 0)
           {
               // Click button 9
```

```
                ((GuiTestObject)tos9[0]).click();
       //unregister
       tos9[0].unregister();
            }
        }
```

```
Public Function TestMain(ByVal args() As Object) As Object

        Dim topWindoprops(3) As Rational.Test.Ft.Script.Property
        Dim btn9props(2) As Rational.Test.Ft.Script.Property
        Dim tos() As TestObject
        Dim tos9() As TestObject
        Dim guiObj9 As GuiTestObject

        ' Find the top-level window of calculator application
        topWindoprops(0) = New Rational.Test.Ft.Script.Property(".processName", "calc.exe")
        topWindoprops(1) = New Rational.Test.Ft.Script.Property(".class", "SciCalc")
        topWindoprops(2) = New Rational.Test.Ft.Script.Property(".name", "Calculator")
        topWindoprops(3) = New Rational.Test.Ft.Script.Property(".text", "Calculator")
        tos = Find(AtChild(topWindoprops))

        If (tos.Length > 0) Then

            'Find button that contains the text 9
            btn9props(0) = New Rational.Test.Ft.Script.Property(".class", "Button")
            btn9props(1) = New Rational.Test.Ft.Script.Property(".name", "9")
            btn9props(2) = New Rational.Test.Ft.Script.Property(".text", "9")
            'There is only one result, so using 0 as index
            tos9 = tos(0).Find(AtChild(btn9props))

            If (tos9.Length > 0) Then
                'Click button 9
                guiObj9 = CType(tos9(0), GuiTestObject)
                guiObj9.Click()
            End If
        End If
        Return Nothing
    End Function
```

You can use this sample code to verify the number of open browser instances, the state of each browser instance, and the number of open browser tabs in each browser instance:

```
public class BrowserLength extends BrowserLengthHelper
{
    /**
     * Script Name   : BrowserLength
  * Generated      : Mar 2, 2012 6:09:06 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2012/03/02
  * @author      Functional Test User
  */
 public void testMain(Object[] args)
{
   findNumberofBrowser_tab();
   }
 private void findNumberofBrowser_tab() {
```

```
  // TODO Auto-generated method stub
  TestObject[] browsers = RootTestObject.getRootTestObject().find(atChild(".class","Html.HtmlBrowser"));

  System.out.println("No. of browser instances found: "+browsers.length);
  for(int i=0;i<browsers.length;i++){
    sleep(5);
            BrowserTestObject browser = (BrowserTestObject) browsers[i];
            System.out.println("State of the browser instance "+ " is:
 "+browser.getProperty(".readyState").toString());
            TestObject[] t = browser.find(atDescendant(".class", "Html.HtmlBrowser.Tab"));
            System.out.println("No. of Html.HtmlBrowser.Tab found in the browser instance "+ " is:
 "+(t.length-1));

    }
    }
}
```

```
 Public Class BrowserLength
  Inherits BrowserLengthHelper

  Public Function TestMain(ByVal args() As Object) As Object

   findNumberofBrowser_tab()

   Return (Nothing)
  End Function

  Private Sub findNumberofBrowser_tab()

   Dim browsers As TestObject() = RootTestObject.GetRootTestObject().Find(AtChild(".class",
 "Html.HtmlBrowser"))
   LogInfo("No. of browser instances found: " & browsers.Length)
   For i As Integer = 0 To browsers.Length - 1
    Sleep(5)
    Dim browser As BrowserTestObject = DirectCast(browsers(i), BrowserTestObject)
    LogInfo("State of the browser instance is: " & browser.GetProperty(".readyState").ToString())
    Dim t As TestObject() = browser.Find(AtDescendant(".class", "Html.HtmlBrowser.Tab"))

    LogInfo("No. of Html.HtmlBrowser.Tab found in the browser instance is: " & (t.Length - 1))
   Next
  End Sub
 End Class
```

This code returns these results:

Number of browser instances found:

State of the browser instance <instance number> is:

Number of `Html.HtmlBrowser.Tab` values found in the browser instance <instance number> is:

Searching for SAP TestObjects

With Rational® Functional TesterHCL OneTest™ UI, you can locate one or more SAP `TestObjects` matching a specified search criteria, even without using the Object Map.

Rational® Functional TesterHCL OneTest™ UI supports a `RootTestObject` class to represent a global view of the software under test. To enable the SAP application for testing, you invoke the `enableForTesting` method on the `RootTestObject` class. To search globally, you invoke the find method on the `RootTestObject` class. Valid values for the subitem, which is the first argument of the find method, include `atProperty`, `atChild`, `atDescendant`, and `atList`. There are special properties that apply to the `RootTestObject.find`, including the `.processName`, `.processID`, and `.domain` properties. You can use any one of these subitems and properties. For example, to search for the SAP domain, you can use the `atChild` subitem with the `.domain` property set to `SAP`.

Note: See the SAP GUI Script Framework documentation for more information on SAP GUI Runtime Hierarchy.

After the top level SAP Test Object is found and returned, you can use that object to find various objects of SAP GUI runtime hierarchy. For example:

- You can obtain the `SAPGuiApplicationTestObject` class by invoking the `GetApplication` method on the `SAPTopLevelTestObject` class.
- You can obtain the `SAPGuiConnectionTestObject` class by invoking the `GetProperty("Connections")` method on the `SAPGuiApplicationTestObject` class.
- You can obtain the `SAPGuiSessionTestObject` class by invoking the `GetProperty("Sessions")` method on the `SAPGuiConnectionTestObject` class.
- You can obtain the SAP's active window by invoking the `GetProperty("ActiveWindow")` method on the `SAPGuiSessionTestObject` class.

Once you have the active window object, you can use the `GetChildren` method on the main window test object to find and interact with various objects on `GuiMainWindow` method.

Listed below is an example on how you can perform user interactions with objects in the SAP application. This sample code does these actions:

1. Enables the SAP application for testing
2. Returns the SAP test object representing the window
3. Uses this object to find the **Create Role** button whose button name property is set to `btn[48]` on the SAP toolbar
4. Clicks the **Create Role** button

Example:

```
import resources.HandCodingWithEnablementHelper;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;
```

```
/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class HandCodingWithEnablement extends HandCodingWithEnablementHelper
{
 /**
  * Script Name   : HandCodingWithEnablement
  * Generated     : Sep 5, 2006 10:03:51 AM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2006/09/05
  * @author Administrator
  */
 public void testMain(Object[] args)
 {
  // Searching for SAP Test Objects through Scripting

  // This enables SAP to be tested by Rational® Functional TesterHCL OneTest™
                 UI and
  // returns all top-level test objects in the SAP domain
  getRootTestObject().enableForTesting("sapLogon");
  TestObject[] sapApps = getRootTestObject().find(atChild(".domain", "SAP"));

  // Get a handle to the SAP Application from the top-level SAP object
  if(sapApps.length > 0){
   SAPGuiApplicationTestObject theAPP = ((SAPTopLevelTestObject)sapApps[0]).getApplication();
   logInfo("Application Number:" + theAPP.getProperty("Id"));

   // Get a handle to the SAP Connection from the SAP Application Test object
   TestObject[] cons = (TestObject[])theAPP.getProperty("Connections");

   SAPGuiConnectionTestObject con = (SAPGuiConnectionTestObject)cons[0];
   logInfo("Connection Number:" + con.getProperty("Id"));

   // Get a handle to the SAP Session from the SAP Connection Test Object
   TestObject[] sessions = (TestObject[])con.getProperty("Sessions");
   SAPGuiSessionTestObject sess = (SAPGuiSessionTestObject)sessions[0];
   logInfo("Session Number:" + sess.getProperty("Id"));

   // Get a handle to the SAP Main Window from the SAP Session Test Object
   // and iterate over its children till the desired object is found
   SAPTopLevelTestObject mainWnd = (SAPTopLevelTestObject)sess.getProperty("ActiveWindow");

   TestObject[] wndChild = mainWnd.getChildren();
   for (int i=0; i<wndChild.length; i++)
   {
    String name = (String)wndChild[i].getProperty("Name");
    if (name.compareTo("tbar[1]")== 0)
    {
     TestObject[] btn = (TestObject[])wndChild[i].getChildren();
     for (int j = 0; j< btn.length; j++)
     {
      System.out.println("ToolBar Buttons");
      String btnType = (String)btn[j].getProperty("Type");
      if (btnType.compareTo("GuiButton")==0)
      {
```

```
       SAPGuiToggleTestObject button = (SAPGuiToggleTestObject)btn[j];
       String btnName = (String)button.getProperty("Name");
       if (btnName.compareTo("btn[48]")== 0)
       {
        // Click the "Create Role" button ("btn[48]") placed on the toolbar("tbar[1]")
        button.press();
        logInfo("Clicked on the Create Role button");
        break;
       }
      }
     }
    }
   }
  }else{
   logInfo("SAP Application not found");
  }
 }
}
```

If the SAP application is already enabled, then you do not need to enable the SAP application explicitly for testing. Instead, you can use the following code to find the enabled SAP application.

```
DomainTestObject domains[] = getDomains();
 for  (int i =0; i < domains.length; i ++)
 {
  DomainTestObject domain = domains[i];
  String name = (String)domain.getName();
  if (name.compareTo("SAP") == 0)
  {
   // Returns all top-level test objects in the SAP domain
   TestObject[] sapApps = domains[i].getTopObjects();

   // Perform user interactions with the SAP objects
  }
 }
```

You can also adapt the dynamicfind() API to find SAP text objects in a functional test script and perform setText in an SAP text field.

```
public class SAPEditControl extends SAPEditControlHelper {
 /**
  * Script Name : <b>SAPEditControl</b> Generated : <b>Aug 3, 2011 2:29:57
  * PM</b> Description : Functional Test Script Original Host : WinNT Version
  * 5.1 Build 2600 (S)
  *
  * @since 2011/08/03
  * @author Functional Test User
  */
 public void testMain(Object[] args) {
  // Define a set of properties for a control (test object) to be searched
  Property Props[] = new Property[4];
  // property and value
  Props[0] = new Property(".class", "Html.TABLE");
  Props[1] = new Property(".customclass", "SAPEditControl");
  Props[2] = new Property(".id", "WD019E-r");
  Props[3] = new Property(".classIndex", "10");

  try {
```

```
   // Find and store test objects into array
   TestObject Obj[] = getRootTestObject().find(atDescendant(Props));

   // Perform a click action on the very first object.
   ((TextGuiSubitemTestObject) Obj[0]).click();

   // Set a text into SAP Edit Control
   ((TextGuiSubitemTestObject) Obj[0]).setText("ClaimedAmount");

  } catch (Exception ex) {
   ex.printStackTrace();
  } finally {
   //call unregisterAll to clear reference.
   unregisterAll();
  }
 }

}
```

Tracing AJAX requests

You can test AJAX-based applications in two different ways; by setting the Auto Trace option to true or by setting the Auto Trace option to false on the corresponding document object. By setting Auto Trace option to true, you can trace the AJAX requests.

The following example shows how to use the getAjaxPendingRequests and waitForAjaxCompletedRequests. The getAjaxPendingRequeststo method is used to return the number of AJAX pending requests at any given point of time since the first AjaxTraceOn. The waitForAjaxCompletedRequestsTo method is used to wait for the specified number of AJAX requests to be completed. If the argument is not specified, it waits till the pending requests becomes zero.

Note: When the pending requests become zero, ensure that you turn off the Auto Trace option.

```
public class AjaxRequestExample extends AjaxRequestExampleHelper
{
 /**
  * Script Name   : AjaxRequestExample
  * Generated     : Apr 24, 2012 2:52:37 PM
  * Description   : Functional Test Script
  * Original Host : WinNT Version 5.1  Build 2600 (S)
  *
  * @since  2012/04/24
  * @author Functional Test User
  */
 public void testMain(Object[] args)
 {
  ajaxTraceTest();
 }

 public void ajaxTraceTest() {
   startBrowser("Internet Explorer",
  "<AJAX related Web Application URL>");
```

```
  // When is set to true, turn the AJAX Request tracing facility to 'On' at the HTML document level.
  document_convertBuyTransferAnd().setAjaxTrace(true);

  // perform AJAX-related actions.
  text_xvalue().click(atPoint(72, 10));
  list_xfrom().click();
  list_xfrom().click(atText("EUR - Euro"));

  /*
   * ++++++++++Use one of the following three functions:
     a) waitForAjaxPendingRequests() - Wait for all the pending request to be served.
     b) waitForAjaxPendingRequests(5) - Wait for the 5 pending request to come
     c) waitForAjaxCompletedRequests(3) - Wait for 3 completed AJAX requests
   *
   * Just to illustrate we have used a,b and c - in the real use case we expect you to use any one of
 the function.
   */

  // Wait until all pending requests are completed.
  document_convertBuyTransferAnd().waitForAjaxPendingRequests();

  // Number of allowed pending requests, otherwise throw AjaxTimeOutException on timeout
  document_convertBuyTransferAnd().waitForAjaxPendingRequests(5);

  // Wait until the specified count of AJAX requests are completed. Otherwise throw
 AjaxWaitTimeOutException on timeout.
  document_convertBuyTransferAnd().waitForAjaxCompletedRequests(1);

  // When is set to false, turn the AJAX Request tracing facility to off
  document_convertBuyTransferAnd().setAjaxTrace(false);

 }
}
```

Searching for GEF objects

Rational® Functional TesterHCL OneTest™ UI recognizes the GEF EditParts and Palettes. Some Figures may not have an association with an EditPart. You can use Rational® Functional TesterHCL OneTest™ UI APIs to find such Figures as shown in the below examples.

Example 1: The following example shows how to use getFigure() API to retrieve a Figure that has the text "label" and is not associated with EditPart

```
//Get the figure for an EditPart
  GuiTestObject figureTO = EntityEditPart().getFigure();

//Find for a figure that has the text in it.
  TestObject foundTO[] = figureTO.find(atDescendant("text", "label"));
  if(foundTO != null)
  {
   int numFound = foundTO.length;
   for(int index = 0; index < numFound ; index ++)
   {
    if(foundTO[index] != null && foundTO[index] instanceof GuiTestObject)
    {
     //To check for specific property on the figure
```

```
     Object figWidth = foundTO[index].getProperty("width");
     if(figWidth != null)
      ((GuiTestObject)foundTO[index]).click();
    }
   }
  }
```

Example 2: The following example shows how to use getConnectors() API to perform click operation on the connector that has the label "Association"

```
//List the connectors from the node's parent
  TestObject parent = EntityEditPart().getParent();
  if(parent != null && parent instanceof GefEditPartTestObject)
  {
   TestObject connectors[] = ((GefEditPartTestObject)parent).getConnectors();

   if(connectors != null)
   {
    int numConnector = connectors.length;
    for(int conIndex = 0; conIndex < numConnector; conIndex ++)
    {
     if(connectors[conIndex] != null && connectors[conIndex] instanceof GefEditPartTestObject)
     {
      GuiTestObject figConnector = ((GefEditPartTestObject)connectors[conIndex]).getFigure();
      //Find for a figure that has some text in it.
      TestObject foundConn[] = figConnector.find(atDescendant("text", "association"));
      if(foundConn != null && foundConn.length> 0)
      {
       //If there is only one label with the text "Association"
       if(foundConn[0] != null && foundConn[0] instanceof GuiTestObject)
       {
        ((GuiTestObject)foundConn[0]).click();
       }
      }

     }

    }
   }
  }
```

Example 3: The following example populates a list with connectors that are descendants to the selected EditPart using the isConnector () API

```
//Assuming you have "RootEditPart" in the ObjectMap.
 ArrayList connList = new ArrayList();
 enumerateAllConnectors(RootEditPart(),connList);
 }

   private static void enumerateAllConnectors(TestObject editPart,ArrayList connList)
 {
  if(editPart != null )
  {
   if(editPart instanceof GefEditPartTestObject)
   {
    boolean isConnector = ((GefEditPartTestObject)editPart).isConnector();
    if(isConnector)
```

```
     connList.add(editPart);
   }

   TestObject []children = editPart.getChildren();
   if(children != null)
   {
    int numChild = children.length;
    for(int i=0; i < numChild ; i++)
    {
     enumerateAllConnectors(children[i], connList);
    }
   }
  }
 }
```

Passing parameters by using the describe function in PowerBuilder

In PowerBuilder, you can use the `describe()` function to identify the properties of DataWindow objects and their controls. The `describe()` function is available only with the PowerBuilder DataWindow. The `describe()` function returns a string as a result of the parameters that are specified as a part of the `describe()` function. For example, you can find the data types of the column in a table style presentation. You can use the describe function in Rational® Functional TesterHCL OneTest™ UI as shown in the following examples.

Example 1: This example shows how to pass parameters to the PowerBuilder `describe()` function and report the result as a string. The result displays the employee name and the state of origin of the employee.

```
//Get the figure for an EditPart
  public void testMain(Object[] args)
{


String ls_request;
String ls_report;

ls_request = "DataWindow.Bands DataWindow.Objects "
+ "empname_h.Text "
+ "empname_h.Type emp.Type emp.Coltype "
+ "state.Type empname.Type empname_h.Visible";
ls_report = dw_1.Describe(ls_request);

}
```

See the PowerBuilder help for detailed information on PowerBuilders `describe()` function.

Finding the state of the browser

When you record functional test scripts, if you find that some controls were not picked up by the recording, you can verify whether the browser used during the recording was in a ready state for recording. Similarly, if you encountered problems during playback, you can verify the state of the browser. You can use the dynamic find() API and use the Html.HtmlBrowser method for this purpose.

This example shows you how to use the dynamic find() API and use the Html.HtmlBrowser method to verify the state of a browser during recording or playback.

Note: This example assumes a single instance of the browser. You can use this example iteratively when multiple instances of the browser are running.

```
public void testMain(Object[] args)
{
 //This sample verifies whether the Browser is in ready state or not.
 // To run this script, start a single instance of the browser, Internet Explorer or Mozilla Firefox.

 startBrowser("http://www.google.com");
 sleep(5);
 // Checking Browser class and when it is found, returns to Test Object
 TestObject[]  to = find(atChild(".class", "Html.HtmlBrowser"));
 // Found one or more Test Object
 if(to.length> 0)
 {
  // Cast into BrowserTestObject
  BrowserTestObject bto = (BrowserTestObject)to[0];

  //Wait for the browser to be ready
  // parameter, browsser test object, state of the browser, timeout& delay in seconds
  boolean isBrowserReady = waitForBrowserTobeReady(bto, 4, 240, 10);
  if(isBrowserReady)
  {

   // Performing a find operation and saving the returned object in the TestObject array.
   TestObject[] googleButton =  bto.find(atDescendant(".class" ,"Html.INPUT.submit",".value","I'm
Feeling Lucky"));

   if(googleButton.length ==0 )
   {
    System.out.println("None found");
    return;
   }
   //Click the first test object found.
   ((GuiTestObject)googleButton[0]).click();

  }
  else
  {
   System.out.println("Browser didn't come to ready State");
  }
  unregisterAll();


 }
 else
 {
  System.out.println("No browser instance found");
 }
}

/*
```

```
 *
 * waitForBrowserTobeReady
 * param :
 * This method waits for the browser to come to the readyState within a specified time range
 * BrowserTestObject as bto
 * readyState as 4
 * timeout as 120 seconds
 * delay as 10 seconds
 */

static boolean waitForBrowserTobeReady(BrowserTestObject bto, int readyState, int timeout, int delay)
{
 //Check is browser is ready
 boolean isBrowserReady = false;

 // Number of tries with a delay
 int noOfTries = timeout/delay;

 for(int i=0; i < noOfTries; i++)
 {
  try
  {

   //Possible .readyState property values for the browser
   // 0 - Uninitalized
   //1,2 - LOADING
   //3 - LOADED
   //4 - ENABLE/VISIBLE/READY
   int browserState = ((Integer)(bto.getProperty(".readyState"))).intValue();
   if(browserState>= readyState)
   {
    isBrowserReady = true;
    break;
   }
  }
  //Catch exception if any
  catch(Exception e)
  {
   break;
  }
  sleep(delay);
 }
 //Return successful of browser ready state is true
 return isBrowserReady;
}
}
```

```
    Public Function TestMain(ByVal args() As Object) As Object
        'This sample verifies whether the Browser is in ready state or not.
        ' To run this script, start a single instance of the browser, Internet Explorer or Mozilla
Firefox.
        StartBrowser("http://www.google.com")
        Sleep(5)
        ' Checking Browser class and when it is found, returns to Test Object
        Dim [to] As TestObject() = Find(AtChild(".class", "Html.HtmlBrowser"))
        ' Found one or more Test Object
        If [to].Length> 0 Then
            ' Cast into BrowserTestObject
```

```
            Dim bto As BrowserTestObject = DirectCast([to](0), BrowserTestObject)

            'Wait for the browser to be ready
            ' parameter, browsser test object, state of the browser, timeout& delay in seconds
            Dim isBrowserReady As Boolean = WaitForBrowserTobeReady(bto, 4, 24, 1)
            If isBrowserReady Then

                ' Performing a find operation and saving the returned object in the TestObject array.
                Dim googleButton As TestObject() = bto.Find(AtDescendant(".class", "Html.INPUT.submit",
".value", "I'm Feeling Lucky"))

                If googleButton.Length = 0 Then
                    LogInfo("None found")
                    Return Nothing
                End If
                'Click the first test object found.

                DirectCast(googleButton(0), GuiTestObject).Click()
            Else
                LogInfo("Browser didn't come to ready State")
            End If


            UnregisterAll()
        Else
            LogInfo("No browser instance found")
        End If
        Return Nothing
    End Function

    '
    '  *
    '  * waitForBrowserTobeReady
    '  * param :
    '  * This method waits for the browser to come to the readyState within a specified time range
    '  * BrowserTestObject as bto
    '  * readyState as 4
    '  * timeout as 120 seconds
    '  * delay as 10 seconds
    '
    Private Shared Function WaitForBrowserTobeReady(ByVal bto As BrowserTestObject, ByVal readyState As
Integer, ByVal timeout As Integer, ByVal delay As Integer) As Boolean
        'Check is browser is ready
        Dim isBrowserReady As Boolean = False

        ' Number of tries with a delay
        Dim noOfTries As Integer = timeout / delay

        For i As Integer = 0 To noOfTries - 1
            Try
                'Possible .readyState property values for the browser
                ' 0 - Uninitalized
                '1,2 - LOADING
                '3 - LOADED
                '4 - ENABLE/VISIBLE/READY
                Dim browserState As Integer = DirectCast(bto.GetProperty(".readyState"), Integer)
```

```
                If browserState>= readyState Then
                    isBrowserReady = True
                    Exit For
                End If
                'Catch exception if any
            Catch e As System.Exception
                Exit Try
            End Try
            Sleep(delay)
        Next
        'Return successful of browser ready state is true
        Return isBrowserReady
    End Function
```

Finding objects in a Dojo tree

You can use the dynamic find() API and the dojoTreeExpand() method to find all objects within a Dojo tree control within the application under test.

This example shows you how to use DojoTreeTestObject() to find all objects within a Dojo tree. You can adapt the code to change the browser if required.

```
public void testMain(Object[] args) {
 // TODO Insert code here
 dojoTreeExpand();
}

public void dojoTreeExpand() {

 //Bring the application under test dynamically using startBrowser method and browser as Mozilla
Firefox (assuming Firefox is enabled correctly)
 //Tips: You change the browser to Internet Explorer in the startBrowser method.
 ProcessTestObject process = RationalTestScript.startBrowser(
   "Mozilla Firefox", "http://docs.dojocampus.org/dijit/Tree" (http://docs.dojocampus.org/dijit/Tree));
 // ProcessTestObject process =
 // RationalTestScript.startBrowser("Internet Explorer",
 // "http://docs.dojocampus.org/dijit/Tree" (http://docs.dojocampus.org/dijit/Tree));

 // Wait for the browser to load completely.
 process.waitForExistence();

 // The RootTestObject represents a global view of the Application being
 // tested. It does not
 // represent an actual TestObject in the software under test. it
 // provides ways to finding an arbitrary
 // TestObject based on properties
 RootTestObject to = RootTestObject.getRootTestObject();

 // Define Test Object array
 TestObject[] dojoControls = null;
 for (int i = 0; i <= 10; i++) {

  // Performing a find operation and saving the returned object in the TestObject
  // array.
  dojoControls = to.find(RationalTestScript.atDescendant(".class",
    "Html.A", ".className", "show"));
```

```
   if (dojoControls.length>= 1) {
    break;
   }
   RationalTestScript.sleep(3);
  }
  // Assigning the first found Test Object to the GUITestObject, and
  // perform a click
  ((GuiTestObject) dojoControls[0]).click();

  //Wait enough to load the page completly.
  sleep(30);

  // Define Test Object array, for a Dojo Tree structure
  TestObject[] trees = null;
  for (int i = 0; i <= 10; i++) {
   // Doing a find operation and saving the returned object in the TestObject
   // array.
   trees = to.find(RationalTestScript.atDescendant(".dojoclass",
     "tree", ".id", "treeOne"));
   if (trees.length == 1) {
    break;
   }
   RationalTestScript.sleep(3);
  }
  //
  DojoTreeTestObject dijitTree = new DojoTreeTestObject(trees[0]);

  // Dispatched when a tree has 'Continents' as node is expanded
  dijitTree.expand(atList(atText("Continents"), atText("North America"),
    atText("Mexico")));
  // Dispatched when a tree has 'Continents' as node is expanded
  dijitTree.click(atList(atText("Continents"), atText("North America"),
    atText("Mexico"), atText("Guadalajara")));
  sleep(10);
 }
}
```

```
    Public Function TestMain(ByVal args() As Object) As Object
        dojoTreeExpand()
        Return Nothing
    End Function

    Public Sub dojoTreeExpand()

        'Bring the application under test dynamically using startBrowser method and browser as Mozilla
 Firefox (assuming Firefox is enabled correctly)
        'Tips: You change the browser to Internet Explorer in the startBrowser method.
        Dim process As ProcessTestObject = RationalTestScript.StartBrowser("Mozilla Firefox",
 "http://docs.dojocampus.org/dijit/Tree")

        ' Wait for the browser to load completely.
        process.WaitForExistence()

        ' The RootTestObject represents a global view of the Application being
        ' tested. It does not
        ' represent an actual TestObject in the software under test. it
        ' provides ways to finding an arbitrary
        ' TestObject based on properties
```

```
        Dim [to] As RootTestObject = RootTestObject.GetRootTestObject()

        ' Define Test Object array
        Dim dojoControls As TestObject() = Nothing
        For i As Integer = 0 To 10

            ' Performing a find operation and saving the returned object in the TestObject
            ' array.
            dojoControls = [to].Find(RationalTestScript.AtDescendant(".class", "Html.A", ".className",
"show"))
            If dojoControls.Length>= 1 Then
                Exit For
            End If
            RationalTestScript.Sleep(3)
        Next
        ' Assigning the first found Test Object to the GUITestObject, and
        ' perform a click
        DirectCast(dojoControls(0), GuiTestObject).Click()

        'Wait enough to load the page completly.
        Sleep(30)

        ' Define Test Object array, for a Dojo Tree structure
        Dim trees As TestObject() = Nothing
        For i As Integer = 0 To 10
            ' Doing a find operation and saving the returned object in the TestObject
            ' array.
            trees = [to].Find(RationalTestScript.AtDescendant(".dojoclass", "tree", ".id", "treeOne"))
            If trees.Length = 1 Then
                Exit For
            End If
            RationalTestScript.Sleep(3)
        Next
        '
        Dim dijitTree As New DojoTreeTestObject(trees(0))

        ' Dispatched when a tree has 'Continents' as node is expanded
        dijitTree.expand(AtList(AtText("Continents"), AtText("North America"), AtText("Mexico")))
        ' Dispatched when a tree has 'Continents' as node is expanded
        dijitTree.click(AtList(AtText("Continents"), AtText("North America"), AtText("Mexico"),
AtText("Guadalajara")))
        Sleep(10)
    End Sub
```

Reading multiple datasets from a functional test script

You can use the dynamic find() API to read multiple datasets from a functional test script.

This sample code shows you how to read more than one dataset from a functional test script.

```
public class UserInformation extends UserInformationHelper {
 /**
  * Script Name : UserInformation Generated : Sep 6, 2011 3:57:48
  * PM Description : Functional Test Script Original Host : WinNT Version
  * 5.1 Build 2600 (S)
  *
  * @since 2011/09/06
```

```
 * @author user1
 */
public void testMain(Object[] args) throws Exception {

 //User defined function to load more then on dataset
 firstdataset();
}


public void firstdataset(){

 //Get a value from the first dataset at the Test Script Level.
 String address = dpString("Address");

 System.out.println(" -- Address from the 'script' associated dataset: " + address);

 //Call the second dataset
 Seconddataset();

}

public void Seconddataset() {


 // Point to the dataset location that was created
 java.io.File dpFile = new java.io.File(
   (String) getOption(IOptionName.DATASTORE), "/UserDetails.rftdp");
 // Load the dataset using FT IdatasetFactory
 Idataset dataset_two = dpFactory().load(dpFile, true);

 // Open the dataset using FT IdatasetFactory
 IdatasetIterator dataset_Ite_2 = dpFactory().open(dataset_two, null);

 // After it is opened, initilize the dataset to access the data
 dataset_Ite_2.dpInitialize(dataset_two);

 // Get a value from the second dataset, first record
 String firstName = dataset_Ite_2.dpString("FirstName");

 // Redirect the output to console or use logInfo method
 System.out.println(" -- First Name from the Second dataset: "
   + firstName);
}

}
```

```
Public Class UserInformation
 Inherits UserInformationHelper
 '*
 '     * Script Name : UserInformation Generated : Sep 6, 2011 3:57:48
 '     * PM Description : Functional Test Script Original Host : WinNT Version
 '     * 5.1 Build 2600 (S)
 '     *
 '     * @since 2011/09/06
 '     * @author IBM Rational
 '
```

```
 Public Function TestMain(ByVal args() As Object) As Object
  firstdataset()
  Return Nothing
 End Function


 Public Sub firstdataset()

  'Get a value from the first dataset at the Test Script Level.
  Dim address As String = DpString("Address")

  LogInfo(" -- Address from the 'script' associated dataset: " + address)

  'Call the second dataset
  Seconddataset()

 End Sub

 Public Sub Seconddataset()


  ' Point to the dataset location that was created
  Dim dpFile As New System.IO.FileInfo(DirectCast(GetOption(IOptionName.DATASTORE), String) +
"/UserDetails.rftdp")
  ' Load the dataset using RFT IdatasetFactory
  Dim dataset_two As Idataset = DpFactory().Load(dpFile, True)

  ' Open the dataset using RFT IdatasetFactory
  Dim dataset_Ite_2 As IdatasetIterator = DpFactory().Open(dataset_two, Nothing)

  ' After it is opened, initilize the dataset to access the data
  dataset_Ite_2.DpInitialize(dataset_two)

  ' Get a value from the second dataset, first record
  Dim firstName As String = dataset_Ite_2.DpString("FirstName")

  ' Redirect the output to console or use logInfo method
  LogInfo(" -- First Name from the Second dataset: " + firstName)
 End Sub

End Class
```

Selecting an item from a Java drop-down list

You can modify the dynamic find() API to use the mapped and dynamic test objects, to select an item from a drop-down list in a Java application, as the following example illustrates:

This example uses the Classics Java application that is provided with Rational® Functional TesterHCL OneTest™ UI.

```
//Start the Classics JavaA application provided with Rational® Functional TesterHCL OneTest™
                  UI
        startApp("ClassicsJavaA");

        // Frame: ClassicsCD - Click the Place Order button (mapped test object)
        placeOrder().click();
```

```
   //Define root test objects
       RootTestObject  root = getRootTestObject();

       //Find the Java comboBox using properties exposed by the comboBox, define as array.
       TestObject[] to = root.
     find(atDescendant(".class","javax.swing.JComboBox","name", "nameCombo"));

       // Click the very first object that is found in the test object
       ((TextSelectGuiSubitemTestObject) to[0]).click();

       // Select one of the subitems from the drop-down list
       ((TextSelectGuiSubitemTestObject) to[0]).select("Bill Wu");

       // Frame: Member Logon  - Click the Cancel button
       cancel().click();

       // Frame: ClassicsCD - Close the application
       classicsCD(ANY,MAY_EXIT).close();
```

```
'Start the Classics JavaA application provided with Rational® Functional TesterHCL OneTest™
                UI
StartApp("ClassicsJavaA")

' Frame: ClassicsCD - Click the Place Order button (mapped test object)
PlaceOrder().Click()

'Define root test objects
Dim root As RootTestObject = GetRootTestObject()

'Find the Java comboBox using properties exposed by the comboBox, define as array.
Dim [to] As TestObject() = root.Find(AtDescendant(".class", "javax.swing.JComboBox", "name",
"nameCombo"))

' Click the very first object that is found in the test object
DirectCast([to](0), TextSelectGuiSubitemTestObject).Click()

' Select one of the subitems from the drop-down list
DirectCast([to](0), TextSelectGuiSubitemTestObject).[Select]("Bill Wu")

' Frame: Member Logon  - Click the Cancel button
Cancel().Click()

' Frame: ClassicsCD - Close the application
ClassicsCD(ANY, MAY_EXIT).Close()
```

Verifying the status of a radio button or check box

You can use the dynamic find() API to verify the status of a radio button or check box during playback, such as whether the control was selected or not.

This example shows you how to verify whether a radio button was selected during playback. When you use this example, if the radio button is found to be cleared, it is identified as not selected and is written to the log. If the radio button is found to be selected, it is clicked during playback.

```
private void checkRadionButtonStatus() {

State curState = radioButton_group1Milk().getState(); //radioButton_group1Milk is a mapped test object,
 recorded using the Rational® Functional TesterHCL OneTest™
                 UI recorder.

System.out.println(curState.getState());
// Test whether the radio button is selected.
sleep(2);
if (curState.isNotSelected()) {
logInfo(" -- Html.INPUT.radio Button is NOT selected by default."); // The playback log captures the
 information.

} else {
System.out.println("Selected!!!");
logInfo(" -- HTML Radio Button selected by default,");
//Perform other actions such as click, etc..
}
}
```

```
    Private Sub checkRadionButtonStatus()

        Dim curState As State = RadioButton_group1Milk().getState()
        'radioButton_group1Milk is a mapped test object, recorded using the Rational® Functional
 TesterHCL OneTest™
                 UI recorder.
        LogInfo("State:" & curState.GetState())
        ' Test whether the radio button is selected.
        Sleep(2)
        If curState.IsNotSelected() Then
            ' The playback log captures the information.
            LogInfo(" -- Html.INPUT.radio Button is NOT selected by default.")
        Else
            LogInfo("Selected!!!")
            'Perform other actions such as click, etc..
            LogInfo(" -- HTML Radio Button selected by default,")
        End If
    End Sub
```

This example modifies the previous example to verify the status of an HTML check box, and verify whether it was selected during playback. When you use this example, if the check box is found to be cleared, it is identified as not selected and is written to the log. If the check box is found to be selected, it is clicked during playback.

```
private void checBoxStatus() {

State currentStatus_checkBox = checkBox_ctl00EnrollmentConten().getState();//
 checkBox_ctl00EnrollmentConten is a mapped test object, recorded using the Rational® Functional
 TesterHCL OneTest™
                 UI recorder.

System.out.println(currentStatus_checkBox.getState());
// Test whether the check box is selected.
sleep(2);
if (currentStatus_checkBox.isNotSelected()) {
logInfo(" -- Html.INPUT.checkbox is NOT selected by default.");
} else {
```

```
System.out.println("Selected!!!");
logInfo(" -- Html.INPUT.checkbox is selected by default,"); // The playback log captures the
 information.
//Perform other actions such as click, etc..
}
}
```

```
    Private Sub checBoxStatus()

        Dim currentStatus_checkBox As State = CheckBox_ctl00EnrollmentConten().getState()
        ' checkBox_ctl00EnrollmentConten is a mapped test object, recorded using the Rational®
 Functional TesterHCL OneTest™
                 UI recorder.
        LogInfo("State:" & currentStatus_checkBox.GetState())
        ' Test whether the check box is selected.
        Sleep(2)
        If currentStatus_checkBox.isNotSelected() Then
            LogInfo(" -- Html.INPUT.checkbox is NOT selected by default.")
        Else
            LogInfo("Selected!!!")
            ' The playback log captures the information.
            'Perform other actions such as click, etc..
            LogInfo(" -- Html.INPUT.checkbox is selected by default,")
        End If
    End Sub
```

These examples show how to use the `isNotSelected()` method. You can adapt these examples for your requirements.

Closing active browsers before playback

You can use the dynamic find() API to find and close all active browsers before you play back a script to ensure that there are no active browser instances before playback.

This example shows you how to use the dynamic find() API to search through the active domains, find all active instances of the Microsoft Internet Explorer browser and close them. You can also modify this example to find and close active Mozilla Firefox browsers.

```
public void closeAllBrowsers() {
 DomainTestObject dom[] = getDomains(); // Get all domains
 for (int i = 0; i < dom.length; i++) {
  try {
   String s = (dom[i].getImplementationName()).toString();
   if ("MS Internet Explorer".equals(s)) { // If browser name equals MS Internet Explorer
    (dom[i].getProcess()).kill(); // Shut down the process
    sleep(2);
   }
  } catch (TargetGoneException e) {

  }

  unregisterAll(); // Ensure that you clean up the used test objects to prevent memory-related
problems.
 }
}
```

```
    Public Sub closeAllBrowsers()
        Dim dom As DomainTestObject() = GetDomains()
        ' Get all domains
        For i As Integer = 0 To dom.Length - 1
            Try
                Dim s As String = (dom(i).GetImplementationName()).ToString()
                If "MS Internet Explorer".Equals(s) Then
                    ' If browser name equals MS Internet Explorer
                    dom(i).GetProcess().Kill()
                    ' Shut down the process
                    Sleep(2)
                End If

            Catch e As TargetGoneException
            End Try

            ' Ensure that you clean up the used test objects to prevent memory-related problems.
            UnregisterAll()
        Next
    End Sub
```

You can also use the dynamic find() API to directly find and close active instances of active browsers (Microsoft Internet Explorer or Mozilla Firefox), as shown in this example:

```
public void closeAllBrowserUsingfind(){


 // Find browser objects using the Rational® Functional TesterHCL OneTest™
                 UI find function and store into test object
 TestObject[] browsers = find(atChild(".class", "Html.HtmlBrowser"));

 if(browsers.length ==0){
  System.out.println("Found no Html.HtmlBrowser");
  return;
 }

 // Close each browser object found, after casting it to a BrowserTestObject
 for (TestObject browser:browsers) {
     ((com.rational.test.ft.object.interfaces.BrowserTestObject) browser).close();
 }

 // Unregister the test objects.
 unregister(browsers);
}
```

```
    Public Sub closeAllBrowserUsingfind()

        ' Find browser objects using the product find function and store into test object
        Dim browsers As TestObject() = Find(AtChild(".class", "Html.HtmlBrowser"))

        If browsers.Length = 0 Then
            LogInfo("Found no Html.HtmlBrowser")
            Return
        End If

        ' Close each browser object found, after casting it to a BrowserTestObject
        For Each browser As TestObject In browsers
```

```
            DirectCast(browser, Rational.Test.Ft.Object.Interfaces.BrowserTestObject).Close()
        Next

        ' Unregister the test objects.
        Unregister(browsers)
    End Sub
```

The two examples shown above can be used as utility functions in a script helper superclass. For more information about script helper superclasses, see Script helper superclass/base class on page 888.

Closing unexpected HTML dialog boxes during playback

You can use the dynamic find() API to close HTML dialog boxes that are displayed unexpectedly during playback, to ensure smooth playback.

Rational® Functional TesterHCL OneTest™ UI provides the unexpected window handling feature to handle unexpected windows that are displayed during playback. For more information about this feature, see Configuring how to handle unexpected windows during playback on page 1438.

You can also use the dynamic find() API, modified as shown in this example, to close HTML dialog boxes, both Html.Dialog or Html.HtmlDialog, that are displayed unexpectedly during playback. If you use this example during playback, it takes precedence over the unexpected window handling settings in the Configure Handling of Unexpected Windows dialog box.

```
public static void closeHtmlBrowserDialogs() {


 //Get all domains and search each domain
 DomainTestObject domains[] = getDomains();
 for (int i = 0; i < domains.length; ++i) {
  if (domains[i].getName().equals("Html")) { // Look for the HTML web domain. Once HTML is found, get
 the top-level test objects.
   TestObject[] topLevelObjects = domains[i].getTopObjects();
   if (topLevelObjects != null) {
    for (int j = 0; j < topLevelObjects.length; ++j) {

     String className = null;
     try{
      className  = (String) topLevelObjects[j].getProperty(".class");

     } catch (PropertyNotFoundException ex){ // Throw an exception property not found exception
               // when the requested get property was not available
      ex.printStackTrace(); // print the detailed exception if any
     }

     // If Html.HtmlDialog or Html.Dialog pop-up windows are displayed during playback, both are
 handled.
     if((className != null) && (className.equalsIgnoreCase("Html.Dialog")
            ||className.equalsIgnoreCase("Html.HtmlDialog")) )
        ((TopLevelTestObject) topLevelObjects[j]).close(); // The HTML pop-up window is closed.

    }
   }
```

```
 }
}

unregisterAll(); // Clean up any used test objects to prevent memory-related problems.
```

```
   Public Shared Sub CloseHtmlBrowserDialogs()

       'Get all domains and search each domain
       Dim domains As DomainTestObject() = GetDomains()
       For i As Integer = 0 To domains.Length - 1
           If domains(i).GetName().Equals("Html") Then
               ' Look for the HTML web domain. Once HTML is found, get the top-level test objects.
               Dim topLevelObjects As TestObject() = domains(i).GetTopObjects()
               If topLevelObjects IsNot Nothing Then
                   For j As Integer = 0 To topLevelObjects.Length - 1

                       Dim className As String = Nothing
                       Try

                           className = DirectCast(topLevelObjects(j).GetProperty(".class"), String)
                       Catch ex As PropertyNotFoundException
                           ' Throw an exception property not found exception
                           ' when the requested get property was not available
                           LogInfo(ex.Message)
                           ' print the detailed exception if any
                           LogInfo(ex.StackTrace)
                       End Try

                       ' If Html.HtmlDialog or Html.Dialog pop-up windows are displayed during
playback, both are handled.
                       If (className IsNot Nothing) AndAlso (className.Equals("Html.Dialog") OrElse
className.Equals("Html.HtmlDialog") OrElse className.Equals("Html.HtmlBrowser")) Then
                           DirectCast(topLevelObjects(j), TopLevelTestObject).Close()
                           ' The HTML pop-up window is closed.
                       End If
                   Next
               End If
           End If
       Next
       UnregisterAll()
       ' Clean up any used test objects to prevent memory-related problems.
   End Sub
```

Dynamic Find() API for JavaScript dialogs

You can test the JavaScript dialogs when you play back a script by using the Web UI engine. You can use the dynamic find() API to find and take action on the controls of the dynamic JavaScript dialogs such as alert, prompt, or confirm dialogs.

The following example shows how to use the dynamic find() API to search the web page to find the controls such as the Ok, Cancel, or Confirm buttons on the dynamic JavaScript dialogs that pop up when you run the test.

```
public class FindBased extends FindBasedHelper
{
 public void testMain(Object[] args)
```

```
 {

 startBrowser("edge","http://
civcez229.nonprod.hclpnp.com/aries-web/Samples/javascriptsamples/javascript_popupboxes.html");
  sleep(15);
  RootTestObject root = null;
  TestObject[] testObj = null;

//Alert - OK
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.INPUT.button",".id","alert"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","OK"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

//Confirm - OK
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","confirm"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","OK"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

//Confirm - Cancel
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","confirm"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","Cancel"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

//Prompt - OK
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","prompt"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","OK"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
```

```
  sleep(5);

//Prompt - Cancel
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","prompt"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","Cancel"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

//Prompt - Value - OK
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","prompt"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.Dialog"));
  System.out.println("testObj length : " + testObj.length);
  ((TopLevelTestObject) testObj[0]).inputChars("There");
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","OK"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

//Prompt - Value - Cancel
  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.BUTTON",".id","prompt"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.Dialog"));
  System.out.println("testObj length : " + testObj.length);
  ((TopLevelTestObject) testObj[0]).inputChars("Harry Pottar");
  sleep(5);

  root = getRootTestObject();
  testObj = root.find(atDescendant(".class","Html.DialogButton",".text","Cancel"));
  System.out.println("testObj length : " + testObj.length);
  ((GuiTestObject) testObj[0]).click();
  sleep(5);

 }
}
```

Extending the functionality in the Visual Studio IDE

These topics describe various ways that you can customize Rational® Functional TesterHCL OneTest™ UI or use the application programming interface (API) to extend Rational® Functional TesterHCL OneTest™ UI functionality.

- Changing the default script helper base class
    - Creating a script helper base class
    - Changing the default script helper base class for a project
    - Changing the default script helper base class for a script
- Using the API to edit functional test scripts

Experimental Features

Rational® Functional TesterHCL OneTest™ UI contains a set of pre-releases that enable you to test various experimental features. These experimental features, while still in progress, are introduced early in the release to seek your feedback on its overall functionality and performance. You can play around with these capabilities before they are made available and supported as a general feature in an upcoming release.

Important: Experimental features are not rolled out as part of the general features of Rational® Functional TesterHCL OneTest™ UI and should not be used in a production environment.

To access experimental features, click **Window > Preferences > Test > FT Experimental Features**.

You can enable the following experimental feature in Rational® Functional TesterHCL OneTest™ UI:

- **Enable Mixed Content**: Select this option to record and play back tests for applications that use both HTTP and HTTPS protocols. When you enable mixed content, the test captures content from both the protocols that are used by the application. If not, the test throws an exception for mixed content. For example, if you enable mixed content and record an HTTPS application that has HTTP web links, the test captures content from both the protocols without throwing an exception.

Chapter 8. Test Execution Specialist Guide

This guide describes how to automate the playback of the tests by using different methods such as Docker, IBM Cloud Private, Jenkins and so on. The test results can be pushed to Rational® Test Control PanelHCL® Quality Server.

Configuration of test runs from the UI Test perspective

When you create Web UI, mobile, or Windows tests, you must configure the play back of the recorded tests as test runs from the UI Test perspective before you can view their test results.

Configuration of Web UI test runs

See the different ways to play back Web UI tests.

Prerequisites to running Web UI tests

Before you can run a Web UI test, you must complete the prerequisite tasks.

You can find the following information about the prerequisite tasks:

- Setting a specific browser profile for the playback on page 1236
- Clearing cache, cookies, and history of browsers on page 1237

- For Apple® Safari®: Ensure that you have manually enabled the browser for recording Web UI tests. For instructions, see Enabling the Apple Safari browser to perform Web UI tests on macOS on page 538.
- Starting with 9.1.1, you can run a test that was recorded in Google Chrome Device Mode. This allows you to emulate tests of web apps on mobile devices. See Recording a test with Google Chrome Device Mode on page 568 and Running a test recorded in Google Chrome Device Mode on page 1255.
- Also starting with 9.1.1, you can play back tests in Chrome Headless Mode. This allows you to run tests in an automated testing environment where a visible user interface shell is not required. See Running tests in the headless mode on page 1257.
- Starting with 9.2.1, you can use industry-standard mobile browsers, such as Chrome and Safari, to run Web UI tests for mobile web applications. You can run tests with Chrome on Android devices and emulators and with Safari on iOS devices and simulators.



Note: If you want to run a test on Chrome, you must have installed the appropriate version of Chrome driver for the UI Test Agent. For more information on the Chrome driver, see http://appium.io/docs/en/writing-running-appium/web/chromedriver/ and Troubleshooting issues on page 695.

Additional information about browsers

To use a specific browser profile for the test, see Using alternate browser profile for test playback on page 1236.

You can run a Web UI test in the same web browser that was used for the recording or run the test in other web browsers. You can even run a Web UI test in several browsers simultaneously. You can also run the test as part of keyword execution from IBM® Rational® Quality Manager, as part of compound test, or from the Command Line Interface (CLI).

You can run a test that was recorded in Google Chrome Device Mode. This allows you to emulate tests of web apps on mobile devices. See Recording a test with Google Chrome Device Mode on page 568 and Running a test recorded in Google Chrome Device Mode on page 1255. You can also play back tests in Chrome Headless Mode. This allows you to run tests in an automated testing environment where a visible user interface shell is not required. See Running tests in the headless mode on page 1257.

You can play back the Web UI tests by using browsers such as Microsoft Edge Chromium and Opera browsers on computers that are running on Windows or Mac operating systems.

Note: When you install the Opera browser on your computer, you must select **All users on this computer** in the **Install for** drop-down list. Otherwise, the Opera browser is not displayed in the list of browsers that are available for selection to play back the Web UI tests.

When you run a test, the steps in the test looks for the UI objects over and over again until they get the object or timeout. For example, if the step is to click a button with the name `Submit`, the test will look for the button with that name. If the name of the button changed, the test will keep looking for the button and then timeout. By default, the timeout is set to `10s`. You can modify this value for the steps or at the test level. If you modify the timeout value in the step, that value takes precedence over the timeout value specified at the test level.

Web UI tests that were recorded in Internet Explorer, Firefox or Chrome can be played back with the Microsoft Edge browser, but you cannot record them in Microsoft Edge.

Note: If a test includes an action to open a new browser window from the existing window and the URL or the title of the new browser window changes dynamically at every run time and does not matches with the one in the test, the playback of the test fails. If either the URL or the title of the new browser window is constant, use that in the test and remove the other one. For example, if the URL dynamically changes at run time but the title of the window remains same, ensure that the test only includes the title of the window.

Setting a specific browser profile for the playback

When you play back a Web UI test in a browser, the test is played back in the temporary profile of the browser. You can set a specific user profile, which includes the extensions and settings to use for the test, for the browser in IBM® Rational® Functional TesterHCL OneTest™ UI to play back the Web UI test.

About this task

You can set a specific browser profile path for the following browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Opera

When you set the user profile for a browser, the user profile is used when you run the test from the workbench, command line, UrbanCode™ Deploy, and IBM® Rational® Quality Manager.

Note:

- When you set a specific profile for the Firefox browser, Rational® Functional TesterHCL OneTest™ UI creates a copy of the profile that you set and the tests are played back in the profile copy.
- If the test script modifies the Firefox browser profile during the test run, the profile goes back to the default state after the run completes. The Chrome browser persists the changes that occurred during the test run. The difference in behavior is due to the way each browser manages a profile.

1. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.
2. Click the **Browser** tab.
3. Select the checkbox of the browser for which you want to set the profile path.

   The **Browse** button for the browser is enabled.
4. Click **Browse** to browse and select the user profile for the browser.
5. Click **Apply and Close**.

Results

You have set the user profile path for the selected browser.

Related information

Running a Web UI test on page 1243

Use the Profile Manager to create and remove Firefox profiles

Create a new browser user profile

Clearing cache, cookies, and history of browsers

To optimize the performance of web browsers that are used to play back Web UI tests, you can clear cache, cookies, and history of browsers.

Before you begin

- You must have closed all the instances of the web browser for which you want to clear cache, cookies, and history.
- For Mac operating system, you must have provided full disk access to the applications that are used by Rational® Functional TesterHCL OneTest™ UI by performing the following steps:
    1. Click **System Preferences > Security and Privacy**.
    2. Click **Privacy** tab and then select **Full Disk Access**.
    3. Add the following applications for full disk access:
        - Eclipse: `/Applications/IBM/SDP/Eclipse`
        - Java: `/Applications/IBM/SDP/jdk/Contents/Home/bin/java` and `/Applications/IBM/SDP/jdk/Contents/Home/jre/bin/java`

About this task

When you want to clear the browser data of any browser, you must use a different browser to generate reports.

Restriction:

- You cannot clear the browser data of the Internet Explorer browser.
- You cannot clear the browser data in the following scenarios:
    - When you run Web UI tests in Rational® Test Automation ServerHCL OneTest™ Server
    - When you play back Web UI tests on multiple devices in parallel
    - When you play back AFT tests

1. Go to **Preferences > Test Execution > UI Test Playback**.
2. Click the **Desktop** tab.
3. Select the **Clear Cache** and **Clear history** check boxes in the **Clear browser data** section.

   Alternatively, you can also select the check boxes from the **Run Configuration** dialog box by clicking the **Configure UI Test Playback Preference** link.

   You can clear the browser data of the browsers that are installed in the following operating systems:

Operating System	Supported browsers
Windows	Mozilla Firefox, Google Chrome, Opera, and Microsoft Edge
Linux	Mozilla Firefox and Google Chrome
Mac	Safari , Mozilla Firefox, Google Chrome, Opera, and Microsoft Edge

4. Set the user profile based on browsers.

Note: If you do not set any profile, then the temporary profile of the browser is used.

You must perform the following steps to set the user profile for the Google Chrome, Firefox, and Opera browsers:

a. Do the following tasks in the browser:
   i. Open the browser and enter the following text in the browser URL to find the profile path for each of the browsers:
      - For Google Chrome, enter `Chrome://version`.
      - For Firefox, enter `about:support`.
      - For Opera, enter `about:/`.

   The browser details are displayed.

b. Copy the complete profile path from the **Profile Path** field.
   i. Do the following tasks in Rational® Functional TesterHCL OneTest™ UI:
      1. Click **Windows > Preferences > UI Test Playback > Browser** tab.
      2. Select the checkbox of the browser for which you want to clear the cache, history and cookies, and then enter the profile value in the **User Profile** field.

5. Click **Apply** and **Close**.

Results

The browser cache, cookies, and history are cleared soon after you start the play back of Web UI tests.

Setting or changing the ANDROID_HOME path in Linux operating systems

When you want to run or play back recorded mobile tests by using the UI Test agent in IBM® Rational® Functional TesterHCL OneTest™ UI, you must ensure that the **ANDROID_HOME** path is set correctly in computers that run on Linux.

About this task

You must set up the **ANDROID_HOME** path as an environment variable, if you use the UI Test agent to playback recorded mobile tests for Android applications.

You must set up the environment variable on all computers from which you want to run mobile tests for Android applications.

Note: If you encounter errors about the Android sdk path not found when you attempt to play back mobile tests for Android applications on Windows, you might want to check if the value of the **ANDROID_HOME** environment variable is set correctly or change the value to point to the correct path.

1. Go to `<install_Dir>/UITestAgent` directory and stop the UI Test agent service by running the following command:

   sudo ./UITestAgent.sh stop

2. Modify UI Test agent service (**ui-test-agent.service**) by performing the following steps:
    a. Open the file for editing.
    b. Search for the attribute **ANDROID_HOME**.
    c. Change the value of the attribute **ANDROID_HOME** to *<user-home>/Android/Sdk*.

    Note: You must point to the directory where you installed the Android SDK components. It can be the default directory as `<user-home>/Android/Sdk` or to the directory that contains the Android SDK.

    d. Save and close the file.
3. Run the following command:

    sudo ./UIAgentSetup.sh

Results

You have set or changed the path to the **ANDROID_HOME** environment variable in the Linux operating systems.

What to do next

You can play back the recorded tests by using the UI Test agent on computers that run on Linux.

Setting or changing the ANDROID_HOME path in Mac operating systems

When you want to run or play back recorded mobile tests by using the UI Test agent in IBM® Rational® Functional TesterHCL OneTest™ UI, you must ensure that the **ANDROID_HOME** path is set correctly in computers that run on Mac operating systems.

About this task

You must set up the **ANDROID_HOME** path as an environment variable, if you use the UI Test agent to playback recorded mobile tests for Android applications.

You must set up the environment variable on all computers from which you want to run mobile tests for Android applications.

Note: If you encounter errors about the Android sdk path not found when you attempt to play back mobile tests for Android applications on Windows, you might want to check if the value of the **ANDROID_HOME** environment variable is set correctly or change the value to point to the correct path.

1. Go to `<install_Dir>/UITestAgent` directory and stop the UI Test agent service by running the following command:

   sudo ./UITestAgent.sh stop

2. Modify UI Test agent service (**ui.test.agent.plist**) by performing the following steps:

   a. Open the file for editing.

   b. Search for the attribute **ANDROID_HOME**.

   c. Change the value of the attribute **ANDROID_HOME** to *<user-home>/Android/Sdk*.

      Note: You must point to the directory where you installed the Android SDK components. It can be the default directory as `<user-home>/Android/Sdk` or to the directory that contains the Android SDK.

   d. Save and close the file.

3. Run the following command:

   sudo ./UIAgentSetup.sh

Results

You have set or changed the path to the **ANDROID_HOME** environment variable in Mac operating systems.

What to do next

You can play back the recorded tests by using the UI Test agent on computers that run on Mac operating systems.

Setting or changing the ANDROID_HOME path in Windows operating systems

When you want to run or play back recorded mobile tests by using the UI Test agent in IBM® Rational® Functional TesterHCL OneTest™ UI, you must ensure that the **ANDROID_HOME** path is set correctly in computers that run on Windows operating systems.

About this task

You must set up the **ANDROID_HOME** path as an environment variable, if you use the UI Test agent to playback recorded mobile tests for Android applications.

You must set up the environment variable on all computers from which you want to run mobile tests for Android applications.

Note: If you encounter errors about the Android sdk path not found when you attempt to play back mobile tests for Android applications on Windows, you might want to check if the value of the **ANDROID_HOME** environment variable is set correctly or change the value to point to the correct path.

1. Perform any of the following searches in the **Windows Search** field in the Task bar that depends on the version of Windows that is installed on your computer:
   - Enter `Editing System variables`
   - Enter `View advanced system settings`

   The **Advanced** tab of the **System Properties** is displayed.
2. Click **Environment Variables**.

   The **Environment variables** dialog box is displayed.
3. Select any of the following options to set or change the environment variable:
   - To set the environment variable, go to Step 4 on page 1242.
   - To change the value of the environment variable, go to Step 5 on page 1242.
4. Perform the following steps to set the environment variable:
   a. Click **New** in the **System variables** section.

      The **New System Variable** dialog box is displayed.

   b. Enter `ANDROID_HOME` in the **Variable name** field.

   c. Enter the location where the Android SDK is installed on your computer in the **Variable value** field.

   d. Click **OK**.

      The **ANDROID_HOME** variable is added to the list of the system variables.

   e. Go to Step 6 on page 1243.
5. Perform the following steps to change the value of the environment variable:
   a. Select the variable **ANDROID_HOME** from the **System variables** section, and then click **Edit**.

      The **Edit System Variable** dialog box is displayed.

   b. Change the value of the variable **ANDROID_HOME** to *<user-home>/Android/Sdk*.

Note: You must point to the directory where you installed the Android SDK components. It can be the default directory as `<user-home>/Android/Sdk` or to the directory that contains the Android SDK.

c. Click **OK**.

The value of the **ANDROID_HOME** variable is updated in the list of the system variables.

d. Go to Step 6 on page 1243.

6. Restart the computer.

Results

You have set or changed the path to the **ANDROID_HOME** environment variable in Windows operating systems.

What to do next

You can play back the recorded tests by using the UI Test agent on computers that run on Windows operating systems.

Running a Web UI test

To verify that a web application works as designed, run the test in a browser. Optionally, you can run the test in more than one browser at a time to speed up your test effort. Before running the test, you can choose to use a specific browser profile for the test.

Before you begin

You must have completed the following tasks:

- Recorded a Web UI test. See Recording a Web UI test on page 545.
- Ensured that the Scale and layout in the display settings of the system match with the settings used to create the test.

Note: The display scaling must always be 100%.

- Read and completed the tasks mentioned in Prerequisites to running Web UI tests on page 1235 if they apply to the test that you want to run.
- If the packaged drivers for the Google Chrome, Microsoft Edge, and Opera browsers are not compatible with the browsers installed on your computer, during the playback, a link is provided in the **Run Configuration** dialog. You must click the link to install the appropriate driver and only then continue with the playback of tests.

Important: If you do not resolve the driver incompatibility with the browser, and continue with the test run, the following events can occur.

- The incompatible driver is started in the background, but the browser is not visible to you.
- The test fails to run on the selected browser.
- You cannot start another test run as the incompatible driver still runs in the background.

To resove the issue, you must manually end all instances of the browser drivers that run in the backround, and then install the compatible driver as indicated in the **Run Configuration** dialog before you attempt another test run.

When you play back Web UI tests on a remote computer, you can choose to automatically resolve the browser and driver incompatibility of the Google Chrome, Edge, and Opera browsers by selecting the **Fix the browser driver incompatibility** checkbox.

Only those web browsers that are installed on your computer and supported by the Web UI extension are displayed in the list. To run a web test on a mobile device or emulator, the device must be connected and must be in the passive mode.

About this task

You can run a Web UI test in the same web browser that was used for the recording or run the test in other web browsers. You can even run a Web UI test in several browsers simultaneously. You can also run the test as part of keyword execution from IBM® Rational® Quality Manager, as part of compound test, or from the Command Line Interface (CLI).

1. Double-click the test to open it from the **Test Navigator** view.
2. Click **Run Test** to run an individual test or **Run Compound Test** to run a compound test in the Test editor.
   The **Run Configuration** dialog is displayed.
3. Select the run configuration options as required for the test run.
   The options are described in the following table:

Option	Action
Run on several devices and browsers in parallel	Select this checkbox to view and select the devices and browsers on which to run the test. **Restriction:** Use of multiple displays (monitors) on Mac operating systems to play back recorded tests on any supported browser is not supported.
Run using	Click the displayed browser name in this column to select the web browser on which to run the test.

Option	Action
	If the browser driver is incompatible, then Rational® Functional Tester-HCL OneTest™ UI provides a download link to download the correct version of the driver. After you click the **click here** link, you can view the progress bar and relevant messages that indicate the result of the driver download process. If the download fails, you must manually download the driver and place it in the configuration folder.
Collect data to update test steps	Select this checkbox if it is cleared to apply guided healing.
Automatically update the test at end of the playback	Select this checkbox to apply self-healing if you have selected **Collect data to update test steps** option.

4. Click **Next**.
   The **Advanced playback options** page is displayed.
5. Select the advanced playback options as required for the test run.
   The options are described in the following table:

Option	Action
Global steps delay in milliseconds	Select this checkbox and specify the duration of delay in milliseconds to set the delay between all the steps in the test.
Clear browser data	Select this checkbox to optimize the performance of web browsers.
Private Mode	Select this checkbox to run tests in the private mode.
Launch browser with arguments	Select this checkbox to specify the arguments to be used by the browser, and then enter the arguments in the text box. You can enter multiple arguments separated by a semicolon. For example, -<param1>;-<parama2> **Note:** If you select this checkbox, the option remains selected for the subsequent test runs. You can then clear the checkbox if required.

6. Click **Next**.
   The **Performance Measurement** page is displayed. The options on this page are applicable only for mobile tests and they are disabled for Web UI tests.
7. Click **Finish**.
   The selected web browser opens and the test is played back. Do not perform any action on the web browser while the test is playing back. The statistical and live reports show the live data as the test is played back.

Results

After the test run completes, the unified report, statistical report, and the test log are displayed. When running the test in multiple browsers, a single report is displayed for all browses. To view a functional report, you must generate it manually by right-clicking a report in the **Results** folder and clicking **Generate Functional Test Report**. The **Resources** tab in the statistical report is empty because a Web UI test does not monitor resources.

Related information

Optimizing playback of the test on page 593

Running Web UI tests on BitBar Cloud

You can run Web UI tests on mobile devices that are connected to the BitBar cloud device. After the test run is completed, you can view the test reports in IBM® Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Set up your account to access the BitBar cloud. You must have been issued valid credentials such as the URL of the BitBar cloud instance and an API key to authenticate the connection. This key is available under **My Account > My Integrations > API Access** in BitBar cloud.
- Created one or more Web UI tests.

About this task

BitBar is a cloud testing platform which you can connect from Rational® Functional TesterHCL OneTest™ UI or from a remote agent computer. BitBar supports both iOS and Android applications. You must configure the **BitBar Device Cloud Environment** before you can run the tests.

1. Start Rational® Functional TesterHCL OneTest™ UI
2. Select **UI Test** perspective.
3. Go to **Windows > Preferences > Test > Test Execution > UI Test Playback**.
4. Select the **Mobile Device Cloud** tab.
5. Select the **BitBar host** checkbox to enable the **BitBar Device Cloud Environment**
6. Perform the actions listed in the following table:

Option	Action
BitBar host	Enter the URL of the BitBar cloud instance
API key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .

Option	Action
	Note: When the connection with the BitBar cloud instance is successful, Clicking the **Refresh projects and device groups** enables the Project and Device Group fields.
Project	Select the BitBar project from the drop-down list.
Device group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard. The default value is runTest.

7. Click **Apply and Close**.
8. Open the Web UI test for playback from the **Test Navigator** pane.

   The Test is displayed.

9. Click the **Run Test** icon.

   The **Run Configuration** dialog box is displayed.

10. Select the device on the BitBar cloud for the test playback from the **Run Using** drop-down list.
11. Click **Finish**.

    The test playback is started on the selected device.

Results

You successfully ran the Web UI test on the selected device that is on the BitBar cloud.

After the run, the test results are displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the following details from the unified report:

- The details and screen capture that are associated with each test step. To view these details, you can click each test step in the **Details** pane.
- Android device or iOS device on which the test was played back. To view the device, you can click the icon.
- Test information such as AUT, host computer, and duration of the test. To view such information, you can click the icon.
- All the screen captures in a slide show format. To view the screen captures, you can click the icon.

Running Web UI tests on BrowserStack Cloud

You can run Web UI tests on the BrowserStack Cloud so that you can choose devices from a wide range of Android and iOS devices that are available for testing.

Before you begin

You must have completed the following tasks:

- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Created one or more Web UI tests.

About this task

BrowserStack is a cloud testing platform that you can connect from Rational® Functional TesterHCL OneTest™ UI or a remote agent computer. BrowserStack supports both iOS and Android applications. You can run the tests after you configure **BrowserStack Device Cloud Environment**.

1. Start Rational® Functional TesterHCL OneTest™ UI
2. Select **UI Test** perspective.
3. Go to **Windows > Preferences > Test > Test Execution > UI Test Playback**.
4. Select the **Mobile Device Cloud** tab.
5. Select the **BrowserStack host** checkbox to enable the **BrowserStack Device Cloud Environment**.
6. Perform the actions listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

7. Click **Apply and Close**.
8. Open the Web UI test for playback from the **Test Navigator** pane.

   The Test is displayed.

9. Click the **Run Test** icon.

   The **Run Configuration** dialog is displayed.

10. Select the device on the BrowserStack cloud for the test playback from the **Run Using** drop-down list.
11. Click **Finish**.

    The test playback is started on the selected device.

Results

You ran the Web UI test on multiple mobile devices that are avialble on the BrowserStack Cloud.

What to do next

After the run, the test results are displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

You can view the following details from the unified report:

- The details and screen capture that are associated with each test step. To view these details, you can click each test step in the **Details** pane.
- Android device or iOS device on which the test was played back. To view the device, you can click the icon.
- Test information such as AUT, host computer, and duration of the test. To view such information, you can click the icon.
- All the screen captures that are in a slide-show format. To view the screen captures, you can click the icon.

Running Web UI tests on the pCloudy cloud

You can run Web UI test on Android or iOS devices that are connected to the pCloudy cloud testing platform. After the test run is completed, you can view the test reports in IBM® Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Set up your account to access the pCloudy cloud. You must have been issued valid credentials such as the URL of the pCloudy instance, the user name, and an API key to authenticate the connection.
- Verify that the device you want to test is available and in the released state on the pCloudy cloud.
- Created Web UI tests.

About this task

pCloudy is a cloud testing platform that you can connect from Rational® Functional TesterHCL OneTest™ UI or from a remote agent computer. You can then play back recorded Web UI tests for applications or mobile devices. pCloudy

supports both Android and iOS devices. You must configure the pCloudy device cloud environment in Rational® Functional TesterHCL OneTest™ UI before you can run the tests.

1. Start Rational® Functional TesterHCL OneTest™ UI.
2. Select the **UI Test** perspective.
3. Open the Web UI test for playback from the **Test Navigator** pane.

   The Test is displayed.
4. Go to **Window > Preferences > Test > Test Execution > UI Test Playback**.
5. Click the **Mobile Device Cloud** tab.
6. Select the **pCloudy host** checkbox to enable the options in the **pCloudy Device Cloud Environment** panel.
7. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the URL of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

8. Click **Apply**.

   Note: When the connection with the pCloudy instance is successful, the devices that you configured on pCloudy are displayed in the **Run using** drop-down list.

9. Click **Run Test**.

   The **Run Configuration** dialog is displayed.
10. Select the device on the pCloudy instance for the test playback from the **Run using** drop-down list.
11. Click **Finish**.

    The test playback is started on the selected device.

Results

You successfully ran the Web UI test on the selected device that is on the pCloudy cloud instance.

After the run, the test results are displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the following details from the unified report:

- The details and screen capture that are associated with each test step. To view these details, you can click each test step in the **Details** pane.
- Android device or iOS device on which the test was played back. To view the device, you can click the  icon.
- Test information such as AUT, host computer, and duration of the test. To view such information, you can click the  icon.
- All the screen captures in a slide show format. To view the screen captures, you can click the  icon.

Running Web UI tests on Perfecto mobile cloud

You can run Web UI test on Android or iOS devices that are connected to the Perfecto mobile cloud testing platform. After the test run is completed, you can view the test reports in IBM® Rational® Functional TesterHCL OneTest™ UI.

Before you begin

- You must have Perfecto mobile cloud URL and credentials.
- You must have created one or more Web UI tests that have to be tested on the mobile device.

About this task

When you run a Web UI test in the mobile device cloud, you must ensure that the connection is valid. Before the test execution starts, you can check the connection and make sure that the test runs without any interruption.

1. Start Rational® Functional TesterHCL OneTest™ UI.
2. Select the **UI Test** perspective.
3. Open the Web UI test for playback from the **Test Navigator** pane.

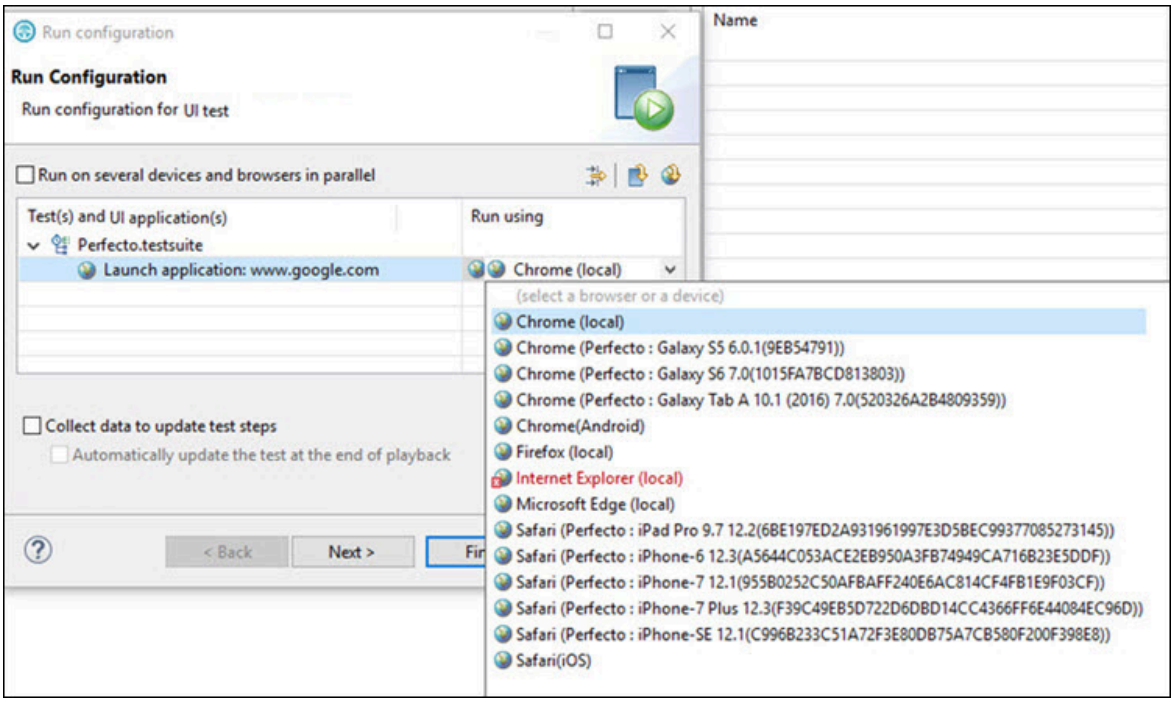
   The Test is displayed.
4. Go to **Window > Preferences > UI Test Playback > Mobile Device Cloud**.
5. Select the **Perfecto Device Cloud Host** check box to enter the details of the Perfecto mobile cloud.
6. Enter the host name in **Perfecto Host**.
7. Enter the API key in **Security Token**.

The credentials are validated and a message is displayed about the connection validity.

8. Click **Apply and Close**.
9. Open the Web UI test for playback from the **Test Navigator** pane.

   The Test is displayed.
10. Click **Run Test**.

The **Run Configuration** dialog displays all the devices along with the name and model number that are available on the Perfecto mobile cloud.

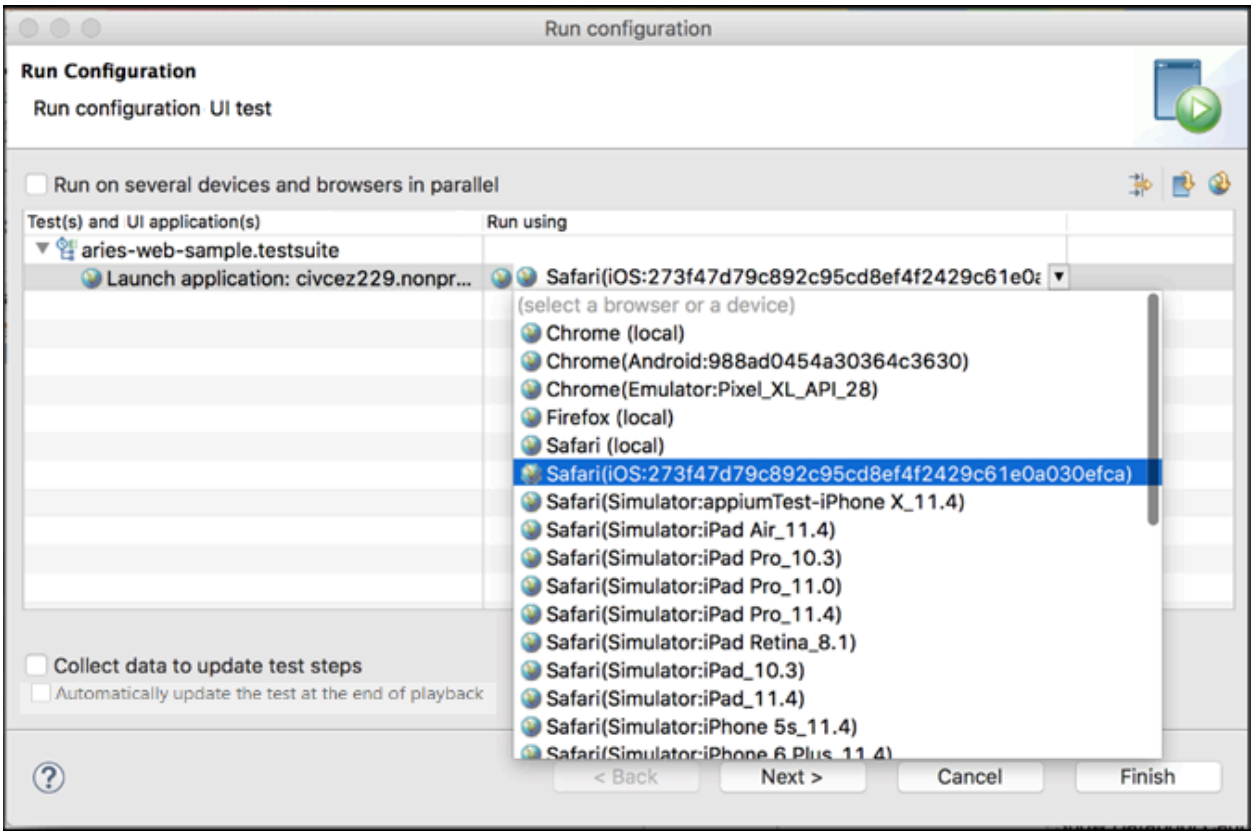


11. Select a device from the **Run Using** list.
12. Click **Finish**.
    The Web UI test starts getting executed on the selected device on the Perfecto mobile cloud.

Running a Web UI test using industry-standard mobile browsers

Starting with 9.2.1, you can use industry-standard mobile browsers, such as Chrome and Safari, to run Web UI tests for mobile web applications. You can run tests with Chrome on Android devices and emulators and with Safari on iOS devices and simulators.

The available devices, emulators, and simulators are listed in the Run wizard.



Here are some of the test execution scenarios that are available to you with this feature:

- You can select connected or configured mobile devices and simulators in the run wizard to run a test. See Running a Web UI test on page 1243.
- You can run a test in parallel on multiple devices. See Running a single Web UI test on multiple browsers and devices simultaneously on page 1376.
- You can run a test on a local computer or on a remote computer by providing details in the **Mobile Devices** tab of the **UI Test Playback** preferences. If these check boxes are enabled, then mobile Chrome or Safari is enabled in the Run Wizard to run the test locally or by using the UI Test agent installed on the remote machine.
    - **Host** - The IP address of the local computer or IP address of the remote computer on which the UI Test Agent is running.
    - **Android Device** - Name of the Android device or emulator. The device name shown by `adb` for a real device or the configured name in `avd` manager for an emulator.

      Note: When you run the UI Test Agent on the remote computer, you can connect to only one Android device or emulator that is running on the remote machine.

    - **iOS Device** - The UDID for the real device or the name of the iOS simulator
    - **Platform Version** - The iOS version of the device

- **Apple Team ID** - The Apple Team ID of the user
- **Role** - The role in the Apple Developer License for the specific registered user

- You can check the connection between the application and mobile device cloud before executing your tests in the mobile cloud by providing the mobile cloud credentials in the **Mobile Device Cloud** tab of the **UI Test Playback** preferences. The credentials are validated and a message is displayed about the connection validity.
  - BitBar host - You must select this check box to enter the details required to connect to the BitBar Cloud.
  - BrowserStack host - You must select this check box to enter the details required to connect to the BrowserStack Cloud.
  - Perfecto host - You must select this check box to enter the details required to connect to the Perfecto Mobile Cloud.
  - pCloudy host - You must select this check box to enter the details required to connect to the pCloudy Cloud.
- You can specify the browser and device details in an XML file to do Accelerated Functional Testing through the command line as described in Running a Web UI test or compound test from the command line on multiple browsers on page 1411.

  Here is a sample XML file for use with Safari and iOS:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits name="smokesuite">
<group>
<tests>
<test path="/WebUIProj/Tests/amazonpixel.testsuite"/>
    </tests>
        <browsers>
              <browser name="safari" id="Simulator:iPhone X"/>

              <browser name="safari" id="iOS:DeviceIdSNo"/>

       </browsers>
    <locations>
            <location host="123.0.0.1" />
   </locations>
  </group>
</inits>
```

  Here is a sample XML file for Chrome on Android devices and emulators:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits name="smokesuite">
<group>
<tests>
<test path="/WebUIProj/Tests/amazonpixel.testsuite"/>
    </tests>
        <browsers>
              <browser name="chrome" id="Emulator:Pixel_XL_API_28"/>

              <browser name="chrome" id="Android:DeviceName"/>

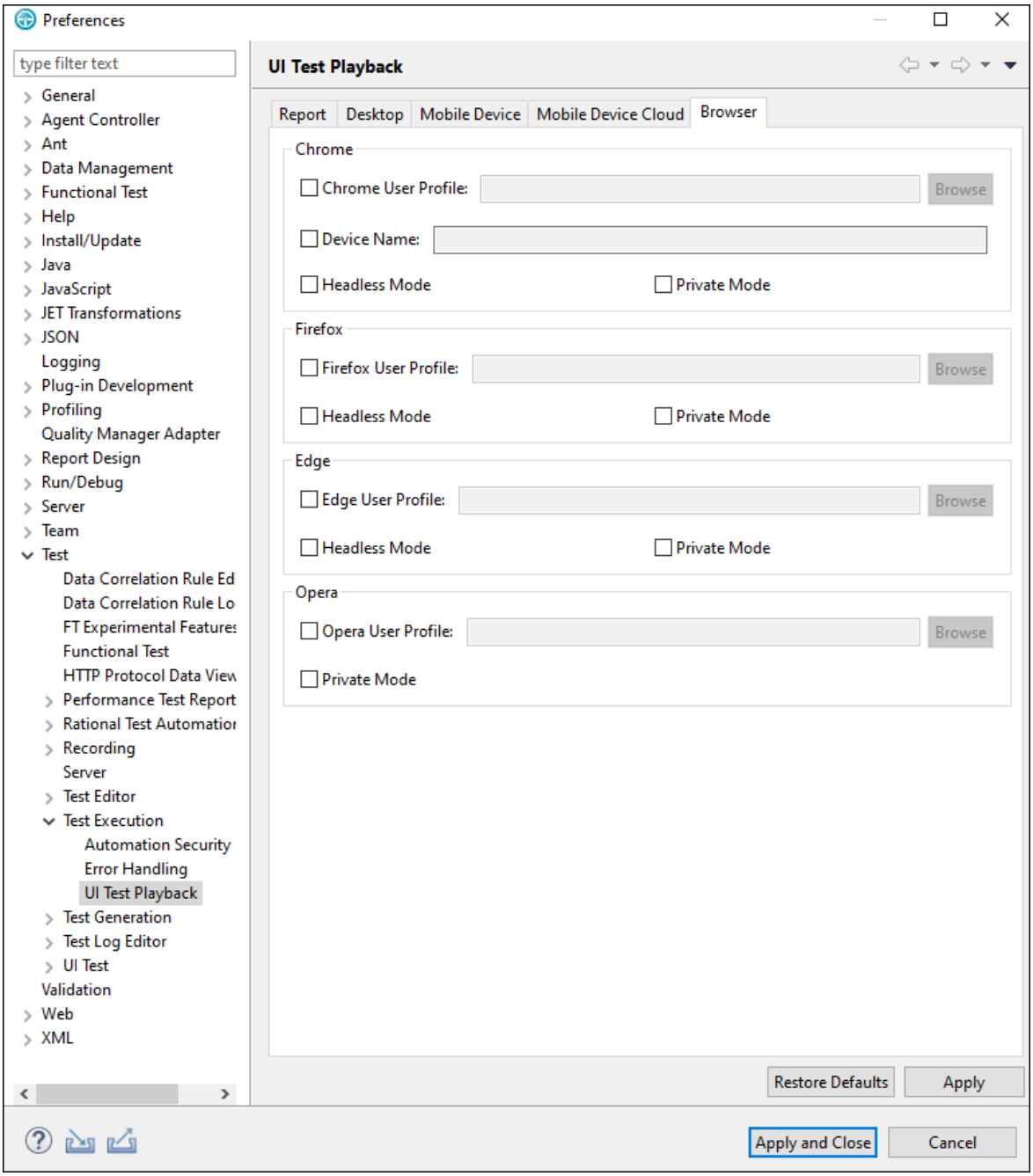
       </browsers>
```

```
    <locations>
            <location host="123.0.0.1" />
   </locations>
  </group>
</inits>
```

Running a test recorded in Google Chrome Device Mode

You can run a test that you recorded in Chrome Device Mode. Doing so allows you to emulate the testing of applications on the browser of a mobile device.

1. Start Rational® Functional TesterHCL OneTest™ UI.
2. Select **Window > Preferences > Test > Test Execution > UI Test Playback**.
   The **Preferences** dialog with the **UI Test Playback** page is displayed.
3. Select the **Browser** tab.



4. Select the **Device Name** checkbox in the **Chrome** section, and then type the name of the device that you used when you recorded the test.

   For example, `iPhone 6` or `Galaxy S5`.

Note: Select the device from the available default list in the browser. Playback on a custom device is not supported. To see the list of devices, see Recording a test with Google Chrome Device Mode on page 568.

5. Click **Apply and Close**.
6. Run the test.
   The device name used for the playback is also viewable in the unified report.

Related information

Unified reports for Web UI, mobile, and Windows tests on page 1456

Running tests in the headless mode

When you do not want to view the GUI of the application under test as the recorded test is played back on a web browser, you can choose to run the test in the headless mode. Running of tests in the headless mode is useful when you use an automated testing environment.

Before you begin

You must have recorded a single Web UI test, compound test with multiple Web UI tests, or an AFT Suite that you want to run in the headless mode.

About this task

The web browsers that support the running of tests in the headless mode are as follows:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox

Note: For information about the supported versions of the web browsers, see System Requirements on page 13.

You can configure a single Web UI test, compound test with multiple Web UI tests, or an AFT Suite to run in the headless mode from Rational® Functional TesterHCL OneTest™ UI or a remote agent computer by performing any of the following actions:

- Select the **Headless Mode** option as a preference for the test playback, if you do not enter the **headless** attribute and its value in the `AFT XML` file of the recorded tests or AFT Suite. You must also select the web browser to be used for the playback.
- Enter the **headless** attribute with its value set to *true* in the `AFT XML` file of the recorded test.

Note: You must have entered the web browser to be used for the playback in the AFT XML file of the recorded tests or AFT Suite.

For example, the entry in the AFT XML file can be as follows:

```
<browser name="chrome" headless="true"/>
```

Notes:

- To ensure that all Web UI tests in an AFT Suite run in the headless mode, you must enter the **headless** attribute with its value set to *true* in the AFT XML file. The tests run in the headless mode irrespective of the preferences configured for the tests in Rational® Functional TesterHCL OneTest™ UI. The Web UI tests also run in the headless mode if you select the **Run Distributed Tests** option for the run.
- If you entered the **headless** attribute with its value set to *false* in the AFT XML file of the AFT Suite, then none of the Web UI tests in the AFT Suite run in the headless mode.

Restriction: The playback of Web UI tests in the headless mode that contain multiple windows is not supported in the Chrome and Edge browsers.

1. Perform any of the following actions:
   - Select the **Headless Mode** option as a preference for the test playback. Go to Step 2 on page 1258.
   - Enter the **headless** attribute in the AFT XML file. Go to Step 3 on page 1258.
2. Start Rational® Functional TesterHCL OneTest™ UI and perform the following steps:
   a. Click **Window > Preferences > Test > Test Execution > UI Test Playback > Browser** tab.
   b. Identify the panel of the web browser that you want to use for the playback test.
   c. Select the **Headless Mode** option for the browser that you want to use to run tests in the headless mode.
   d. Click **Apply and Close**.
   e. Run the test.

   The test is run in the headless mode by using the selected browser.
3. Open the AFT XML file for editing, and then perform the following steps:
   a. Search and find the **browser** element.
   b. Append the **headless** attribute along with the name of the browser that you want to use to the browser element row.

      For example, if you want to run the test in the headless mode by using the Chrome browser, the entry in the AFT XML file can be as follows:

```
<browsers>
<browser name=”chrome” headless=”true” />
</browsers>
```

c. Save and close the AFT XML file.

d. Run the test.

The test is run in the headless mode by using the selected browser.

Related information

Running a Web UI test on page 1243

Running a Web UI test or compound test in the headless mode from the command line on page 1407

Running tests in the private or incognito mode

When you play back Web UI tests, if the user profile that is stored in the browser prevents the test from running without intervention, then you can use the private or incognito mode. Also, when you do not want cookies, caches, and user information to affect your test play back, you can choose the private mode.

Before you begin

You must have completed the following tasks:

- Recorded a Web UI test. See Recording a Web UI test on page 545.
- Verified that the web browser is supported for playing back tests. See System Requirements on page 26.

Restriction: Web UI tests contained in multiple groups in an AFT Suite that are configured to run in the headless mode and in the private mode and grouped based on the browsers are not supported.

About this task

You can configure a single Web UI test, compound test with multiple Web UI tests, or an AFT Suite to run in the private mode on any of the following web browsers:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Opera

To run tests from Rational® Functional TesterHCL OneTest™ UI or a remote agent computer in the private mode, you must select the web browser to be used for the playback, and then select the **Private Mode** as a preference for the test playback.

To run an AFT Suite in the private mode, you must have entered the web browser to be used for the playback and then enter the **private** attribute with its value set to *true* in the AFT XML file of the recorded test.

For example, the entry in the AFT XML file can be as follows:

```
<browser name="chrome" private="true"/>
```

Notes:

- To ensure that all the Web UI tests in an AFT Suite run in the private mode, you must enter the **private** attribute with its value set to *true* in the AFT XML file. The tests run in the private mode irrespective of the preferences configured for the tests in Rational® Functional TesterHCL OneTest™ UI. The Web UI tests also run in the private mode if you select the **Run Distributed Tests** option for the run.
- If you entered the **private** attribute with its value set to *false* in the AFT XML file of the AFT Suite, then none of the Web UI tests in the AFT Suite run in the private mode.

1. Perform any of the following actions:
   - Select the private mode as a preference for the test playback. Go to Step 2 on page 1260.
   - Enter the **private** attribute in the AFT XML file. Go to Step 3 on page 1260.
2. Start Rational® Functional TesterHCL OneTest™ UI and perform the following steps:
   a. Click **Window > Preferences > Test > Test Execution > UI Test Playback > Browser** tab.
   b. Identify the panel of the web browser that you want to use for the playback test.
   c. Select the **Private Mode** option for the browser to run tests in the private mode.
   d. Click **Apply and Close**.
   e. Run the test.

   Note: If you have not set the preference, you can also select the **Private Mode** checkbox under the **Advanced Playback Options** of the **Run Configuration** dialog box when you click **Run Test**.

   The test is run in the private mode by using the selected browser.
3. Open the AFT XML file for editing, and then perform the following steps:
   a. Search and find the browser element.
   b. Append the **private** attribute along with the name of the browser that you want to use to the browser element row.
   You can enter the name of the browsers as indicated in the following table:

Browser	Enter name as
Chrome	chrome
Firefox	firefox
Edge	edge
Opera	opera

c. Save and close the `AFT XML` file.

d. Run the test.

The test is run in the private mode by using the selected browser.

Results

You have run a test in the private mode of a web browser either from Rational® Functional TesterHCL OneTest™ UI or from a remote agent.

What to do next

You can perform any of the following tasks:

- View the test reports. See Running a Web UI test on page 1243.
- Run a Web UI test on multiple browsers. See Running a single Web UI test on multiple browsers and devices simultaneously on page 1376.
- Run multiple Web UI tests in a compound test. See Running multiple Web UI and compound tests simultaneously on page 1357.
- Run an AFT Suite. See Playing back an Accelerated Functional Test asset on page 1356.
- Run an AFT Suite by using the Distributed Tests option. See Accelerating the test effort with distributed testing on page 1352.

Testing with Docker images

IBM® Rational® Functional TesterHCL OneTest™ UI, IBM® Rational® Performance TesterHCL OneTest™ Performance, and IBM® Rational® Performance TesterHCL OneTest™ Performance Agents are available for download as Docker images. You can use them to fulfill the continuous testing aspects of your DevOps lifecycle.

You must use only floating licenses for the product and virtual tester pack floating licenses when playing back tests using Docker. These licenses should be hosted on a server that can be accessed by the workbench.

Configuring Docker containers

You can now deploy and configure the Docker containers on any computer and quickly get started with testing. You can push the product images to the Docker container to automate the playing back of tests.

Before you begin

You must have downloaded and installed Docker. For more information, refer to the following Docker documentation:

- Docker Desktop for Windows
- Docker Engine on Ubuntu
- Other platforms

1. Open PowerShell or a terminal of your choice to run the commands.
2. Run the following command to verify whether your Docker installation was successful:

   ```
   $ docker run hello-world
   ```

3. Download the container image for the agents from the same location that you downloaded the product bits and extract the compressed files.

   Important: The version of the product and agents must match.

4. Run the following command to load the agent image into the Docker repository:

   ```
   tar --wildcards --to-command='docker load' -xzf imageFileName.tar.gz 'images/*'
   ```

   ```
   $ docker load -i imageFileName.tar.gz
   ```

   For example, to load the 10.2.3 Docker image, you can use the following command:

   ```
   tar --wildcards --to-command='docker load' -xzf ibm-rtw-10.2.3.tar.gz 'images/*'
   ```

   ```
   $ docker load -i hcl-onetest-studio-10.2.3.tar.gz
   ```

   When the image is loaded, the following message is displayed:

   ```
   Loaded image: imageFileName:versionNumber
   ```

What to do next

You must set up the playback environment on Docker. See Running tests with containerized agents on page 1262 and Running automated tests with containerized workbench and agents from Docker.

Running tests with containerized agents

When you have a local workbench, instead of installing the agents on different machines and locations, you can deploy the containerized agents to generate the load.

Before you begin

You must have configured the Docker container. See Configuring Docker containers on page 1261.

About this task

Typically, when the agents are installed, you specify the workbench host name and port number to establish the connection with the workbench. But containerized agents are already installed. Therefore, you specify the connection information during the run.

Note: The version number of the container images and the desktop products must match. If you have previous version of the container image, uninstall it, and then install the latest version.

To uninstall the image, you must stop the container by running the `docker stop "CONTAINER ID"` command, and then run the `docker rmi -f "image ID"` command to uninstall the image.

1. Start the container instance of the agent by running the following command:

```
$ docker run -dit --rm -e MASTER_NAME=Workbench_name or IP -e MASTER_PORT=port_number -e
 AGENT_NAME=Agent_name -e AGENT_IP=IP_address imageName:imageVersion
```

Table 54. Description of parameters

Command	Description
-dit	Specifies that the agent container runs in the background.
--rm	Specifies to clean up the container and remove the file system when the container exits.
MASTER_NAME	Specifies the IP or host name of the workbench.
MASTER_PORT	Specifies the port number of the workbench. If you use the default port number of 7080, this command is optional.
AGENT_NAME	Optional: Specifies the name of the agent that report to the workbench.
AGENT_IP	Optional: Specifies the IP address of the agent that report to the workbench.
imageName:imageVersion	Specifies the name and version of the image.

2. Select the Web UI tests to run from the **UI Test** perspective, and then click **Run Distributed Tests** from the context menu.

   The tests are distributed among the connected agents.

Related information

Running multiple Web UI and compound tests simultaneously on page 1357

Running tests in a containerized workbench

You can run tests in a containerized workbench after you deploy your Docker images. You need not install the workbench on your computer to run your tests.

Before you begin

- You must have configured a Docker container.
- You must have exported test assets to a location from where Docker can import them.

Notes:

- You must use a Bash shell to run the commands for executing tests. In Windows operating system, you can use Git Bash.
- The version number of the container images and the desktop products must match. If you have a previous version of the container image, you must uninstall it and then install the current version. To uninstall the image, you must use the following commands:

  To stop the container:

  ```
  docker stop "CONTAINER ID"
  ```

  To uninstall the image:

  ```
  docker rmi -f "image ID"
  ```

Restriction: If you run the test script for an HTML application as part of a compound test and in the headless mode in Docker, the inputKeys action for some special keys might not work. For example, in Firefox, the inputKeys action does not work for arrow keys.

1. Run the following command to load the workbench image into the Docker repository:

   ```
   tar --wildcards --to-command='docker load' -xzf <workbenchImageName> 'images/'*
   ```

   ```
   docker load -i <workbenchImageName>
   ```

   For example, *workbenchImageName* can be hcl-onetest-studio-ibm-rtw-<versionNumber>.tar.gz.

   When the workbench image is loaded into the Docker repository, the following message is displayed:

   ```
   -Loaded image: imageFileName:versionNumber
   ```

2. To run tests on the Docker environment without using any agents, complete the following steps:

   a. Select the required test from the workbench and export it by using the **Test Assets with Dependencies** option. You can use this exported test asset to run on Docker environment.

   b. To run a Web UI test or compound test, start the container by running the following command:

   ```
   $docker run --rm
    -e RATIONAL_LICENSE_FILE=PORT@HOST HCL_ONETEST_LICENSING_URL=<URL> HCL_ONETEST_LICENSIN
   G_ID=<server_ID> -v hostTestAssets:/containerTestAssets -v hostImportedData:/containerImportedData
    -e TEST_IMPORT_PATH=/containerTestAssets/testasset.zip  imageName:imageVersion cmdline
    -workspace /containerImportedData/workspace -project projectName -suite TestSuiteName
   ```

```
 -results autoResults -stdout -exportlog /containerImportedData/testlog.txt
 -exportReport "type=unified;format=<html, xml, or pdf>;folder=runData;filename=report"
```

For example, on Windows host:

```
$docker run --rm -e RATIONAL_LICENSE_FILE=27000@10.116.15.89HCL_ONETEST_LICENSING_URL=
 https://hclsoftware.compliance.flexnetoperations.com
 -e HCL_ONETEST_LICENSING_ID=MYDEMOLICENSEID -v
 C:\TestAssets:/test -v C:\TestExecutions:/runData -e
 TEST_IMPORT_PATH=/test/Project.zip ibm-rtwhcl-onetest-studio:10.0.2 cmdline
 -workspace /runData/workspace -project MyRFTProjectMyHCLOneTestUIProject
 -suite Tests/WebUITest.testsuite -results autoResults
 -stdout -exportlog /runData/logs/WebUItestlog.txt
 -exportReport "type=unified;format=html;folder=runData;filename=WebUIreport"
```

For example, on Linux host:

```
$docker run --rm
 -e RATIONAL_LICENSE_FILE=27000@10.116.15.89HCL_ONETEST_LICENSING_URL=https://
hclsoftware.compliance.flexnetoperations.com -e HCL_ONETEST_LICENSING_ID=MYDEMOLICENSEID
 -v /home/user/TestAssets:/test -v /home/user/TestExecutions:/runData -e
 TEST_IMPORT_PATH=/test/Project.zip ibm-rtwhcl-onetest-studio:10.0.2 cmdline
 -workspace /runData/workspace -project MyRFTProjectMyHCLOneTestUIProject
 -suite "Compound Tests/CompoundTest.testsuite" -results autoResults
 -stdout -exportlog /runData/logs/CompoundTesttestlog.txt
 -exportReport "type=unified;format=html;folder=runData;filename=CompoundTestreport"
```

c. To run an Accelerated Functional Test suite, start the container by running the following command:

```
$docker run --rm -e RATIONAL_LICENSE_FILE=PORT@HOSTHCL_ONETEST_LICENSING_URL=<URL> -e
 HCL_ONETEST_LICENSING_ID=<server_ID> -v hostTestAssets:/containerTestAssets
 -v hostImportedData:/containerImportedData -e TEST_IMPORT_PATH=/containerTestAssets/testasset.zip
  imageName:imageVersion cmdline -workspace /containerImportedData/workspace
 -project projectName -aftsuite AFTTestSuiteName -results autoResults -stdout
 -exportlog /containerImportedData/testlog.txt -exportReport "type=unified;format=<html, xml, or
 pdf>;folder=runData;filename=report"
```

For example:

```
$docker run --rm
 -e RATIONAL_LICENSE_FILE=27000@10.116.15.89HCL_ONETEST_LICENSING_URL=https://
hclsoftware.compliance.flexnetoperations.com -e HCL_ONETEST_LICENSING_ID=MYDEMOLICENSEID
 -v C:\TestAssets:/test -v C:\TestExecutions:/runData -e
 TEST_IMPORT_PATH=/test/Project.ziphcl-onetest-studio ibm-rtwMyHCLOneTestUIProject:10.0.2
 cmdline -workspace /runData/workspace -project MyRFTProject -aftsuite MyAFTSuite.XML
 -results autoResults -stdout -exportlog /runData/logs/AFTSuiteTestlog.txt
 -exportReport "type=unified;format=html;folder=runData;filename=AFTreport"
```

The following table lists the command options with description:

Command	Description
--rm	Removes the container after the run is completed.
-e	Sets the environment variables.
RATIONAL_LICENSE_FILE=<PORT>@<HOST>	Specifies the port number of Rational License Key Server, usually 27000, and the server IP address. The floating

Command	Description
	license for the product and VT-packs must be on the license server.
HCL_ONETEST_LICENSING_URL=<URL>	Specifies the URL of the license server. For example, `https://hclsoftware.compliance.flexnetoperations.com`.
HCL_ONETEST_LICENSING_ID=<ID>	Specifies the cloud license server ID. If you are using a local license server, you must not use this variable. The floating license for the product and VT-packs must be on the license server.
hostTestAssets:/containerTestAssets	Specifies the folder location on the host computer and the container that contains the compressed test assets (.zip format). You must use both the locations to map one or more shared volumes to transfer data such as test assets, logs, and execution results between the host and the container.
hostImportedData:/containerImportedData	Specifies the workspace location on the host computer and the container that contains the test assets that are not compressed. The results of the test execution are saved to the directory you specify on the host computer.
TEST_IMPORT_PATH=<PATH>	Specifies the location of the compressed test assets to be imported into the container. The location path is on the container side and not the host. For example, `/containerTestAssets/archiveName.zip`. The volume and path names are user-defined and must be consistent.
imageName:imageVersion	Specifies the name of the image and its version to run.
cmdline	Specifies the existing command-line arguments to define the following details: ◦ Location of the workspace ◦ Project name ◦ Test name ◦ Results file name ◦ Location of the exported logs ◦ Unified report format ◦ Location of the unified report ◦ Unified report file name

3. Run the following command to view the unified report:

```
docker run -e RATIONAL_LICENSE_FILE=PORT@HOSTHCL_ONETEST_LICENSING_URL=<URL>
 -e  HCL_ONETEST_LICENSING_ID=<server_ID> -v hostTestAssets:/containerTestAssets
 -v hostImportedData:/containerImportedData -p <host_port>:<container_port> -e
 TEST_IMPORT_PATH=/containerTestAssets/testasset.zip imageName:imageVersion cmdline
 -workspace/containerImportedData/workspace -project projectName -suite TestSuiteName
 -results autoResults -stdout -exportlog /containerImportedData/testlog.txt -exportReport
 "type=unified;format=<html, xml, or pdf>;folder=runData;filename=report"
```

For example,

```
docker run -e RATIONAL_LICENSE_FILE=27000@10.116.15.89 -v C:\dockertests\Test:/test
 -v C:\dockertests\Test:/runData -p 7878:7080 -e TEST_IMPORT_PATH=/test/Google1.zip
 ibm-rtw:10.5.1 cmdline -workspace /runData/sandbox1 -project sf -suite test.testsuite
 -results autoResults -stdout -exportlog /runData/logs/WebUItestlog.txt -exportReport
 "type=unified;format=pdf;folder=/runData/logs/Exportedreport1051;filename=testreport"
```

```
docker run -e HCL_ONETEST_LICENSING_URL=https://hclsoftware.compliance.flexnetoperations.com
 -e HCL_ONETEST_LICENSING_ID=MYDEMOLICENSEID -v C:\dockertests\Test:/test -v
 C:\dockertests\Test:/runData -p 7878:7080 -e TEST_IMPORT_PATH=/test/Google1.zip
 hcl-onetest-studio:10.5.1 cmdline -workspace /runData/sandbox1 -project sf -suite test.testsuite
 -results autoResults -stdout -exportlog /runData/logs/WebUItestlog.txt -exportReport
 "type=unified;format=pdf;folder=/runData/logs/Exportedreport1051;filename=testreport"
```

Where, 7878 is the recommended host port and 7080 is the recommended container port.

Sample command output on console:

```

Info : WebdriverUtils - findDownloadDirPath - downloadDrivePath set
 to /etc/opt/IBM/Rational/RFT/configuration

getValue:cmptest1_1680859555198.stats Persistance File : cmptest1_1680859555198.stats
To view the report for the duration of the execution, please open
 http://127.0.0.1:7878/results/ftwebreport/index.html?path=/ftreport/L3J1bkRhdGEvd29ya3NwYWNlL0hD
TF9XZWJVSV9OYXZhbml0X0xvbmdSdW4gX0NvbXBUZXN0X05vRFNfQ291bnQvY21wdGVzdDFfMTY4MDg1OTU1NTE5OC54bW9lY
nJlcG9ydA== in a browser.
Please open a desktop client to view the report after playback is completed.
```

```
INFO: License status: OK
Info : WebdriverUtils - findDownloadDirPath - downloadDrivePath set
 to /etc/opt/HCL/FunctionalTester/configuration
getValue:autoResults_1676021430617.stats Persistance File : autoResults_1676021430617.stats
To view the report for the duration of the execution, please open
 http://127.0.0.1:7878/results/ftwebreport/index.html?path=/ftreport/L3J1bkRhdGEvc2FuZGJveDEvc2Yv
YXV0b1Jlc3VsdHNfMTY3NjAyMTQzMDYxNy54bW9lYnJlcG9ydA== in a browser.
Please open a desktop client to view the report after playback is completed.
```

You can view the live report by using the URL displayed in the command output on console.

For example, http://127.0.0.1:7878/results/ftwebreport/index.html?path=/ftreport/
L3J1bkRhdGEvd29ya3NwYWNlL0hDTF9XZWJVSV9OYXZhbml0X0xvbmdSdW4gX0NvbXBUZXN0X05vRFNfQ291bnQvY21wdGVz
results/ftwebreport/index.html?path=/ftreport/
L3J1bkRhdGEvc2FuZGJveDEvc2YvYXV0b1Jlc3VsdHNfMTY3NjAyMTQzMDYxNy54bW9lYnJlcG9ydA==.

What to do next

When the test run is completed, you can check *hostImportedData* in the host computer to view the exported log.

Related information

Configuring Docker containers on page 1261

Copying test assets with dependencies

Running an AFT suite in a containerized workbench and agents by using Docker Compose

You can run an Accelerated Functional Test (AFT) suite in a containerized workbench and agents after you deploy your Docker images. You can use the Docker Compose tool to run an AFT suite and need not install the workbench or the agents on different computers.

Before you begin

- You must have configured a Docker container.
- You must have exported test assets to a location from where Docker can import them.

Notes:

- You must use a Bash shell to run the commands for executing tests. In Windows operating system, you can use Git Bash.
- The version number of the container images and the desktop products must match. If you have a previous version of the container image, you must uninstall it and then install the current version. To uninstall the image, you must use the following commands:

  To stop the container:

  ```
  docker stop "CONTAINER ID"
  ```

  To uninstall the image:

  ```
  docker rmi -f "image ID"
  ```

Restriction: If you run the test script for an HTML application as part of a compound test and in the headless mode in Docker, the inputKeys action for some special keys might not work. For example, in Firefox, the inputKeys action does not work for arrow keys.

1. Run the following command to load the workbench image into the Docker repository:

   ```
   tar --wildcards --to-command='docker load' -xzf <workbenchImageName> 'images/'*
   ```

   ```
   docker load -i <workbenchImageName>
   ```

   For example, *workbenchImageName* can be hcl-onetest-studio-ibm-rtw-<versionNumber>.tar.gz.

When the workbench image is loaded into the Docker repository, the following message is displayed:

```
-Loaded image: imageFileName:versionNumber
```

2. To run the tests on containerized agents, you must load the agent image into the Docker repository by running the following command:

```
docker load -i <agentImageName>
```

For example, the agent image name could be hcl-onetest-agent-ibm-rtw-<versionNumber>.tar.gz.

When the agent image is loaded into the Docker repository, the following message is displayed:

```
- Loaded image: imageFileName:versionNumber
```

3. From the workbench, create an AFT Suite that lists all the agents that are used for the distributed run of multiple tests on multiple agent locations.
4. To run an AFT Suite, you must initiate a run in which the Docker container agents are automatically connected to the workbench container. To do this, install the Docker Compose tool and complete the following steps:

    a. Create `docker-compose.yml` according to the following sample:



Sample compose file:

```
#SIMPLE DOCKER COMPOSE FILE/TEMPLATE
#BE SURE TO REPLACE ANY PROJECT-SPECIFIC NAMES/PATHS AND LICENSING VARIABLES WITH
 YOUR OWN VALUES
version: '2'
services:
  agent1:
    image: <agentImageName>:<imageVersion>
    environment:
      - MASTER_NAME=<workbenchImageName>
      - AGENT_NAME=<agentImageName>

  agent2:
    image: <agentImageName>:<imageVersion>
    environment:
      - MASTER_NAME=<workbenchImageName>
      - AGENT_NAME=<agentImageName-2>

  workbench:
    image: <workbenchImageName>:<imageVersion>
    entrypoint: cmdline -workspace /runData/workspace -project projectName
-aftsuite AFTSuiteName -results autoResults -stdout -exportlog /runData/agentlog.txt
-exportReport "type=unified;format=html;folder=rundata;filename=report"
    ports:
         - "7080:7080"
         - "7443:7443"
    volumes:
      - C:\Tests:/Tests
      - C:\runData:/runData
    environment:
      - RATIONAL_LICENSE_FILE=27000@<IP> - HCL_ONETEST_LICENSING_URL=<URL>
```

```
      - HCL_ONETEST_LICENSING_ID=<ID>
      - TEST_IMPORT_PATH=/Tests/Project.zip
```

Notes:

- You must replace the values in italics in the sample compose file with values according to your environment.
- Docker Compose tool is included with some versions of Docker. The tool automates some network configurations and makes it easier to coordinate multiple containers. To check whether you already have Docker Compose, you can run the command, `docker-compose --version`.

5. Verify whether the run is completed successfully. If you have used an option such as -exportlog to generate results to a shared volume, check the directory in your host computer that was mapped to hostImportedData and retrieve the exported data.
6. To stop the agents when the workbench container exits, run the following command:

```
docker-compose up --abort-on-container-exit
```

Table 55.

Command	Description
MASTER_NAME	Specifies the name of the workbench image.
AGENT_NAME	Specifies the name of the agent image.
RATIONAL_LICENSE_FILE=<PORT>@<HOST>	Specifies the port number of Rational License Key Server, usually 27000, and the server IP address. The floating license for the product and VT-packs must be on the license server.
HCL_ONETEST_LICENSING_URL=<URL>	Specifies the URL of the license server. For example, `https://hclsoftware.compliance.flexnetoperations.com`.
HCL_ONETEST_LICENSING_ID=<ID>	Specifies the cloud license server ID. If you are using a local license server, you must not use this variable. The floating license for the product and VT-packs must be on the license server.
hostImportedData:/containerImportedData	Specifies the workspace location on the host computer and the container that contains the test assets that are not compressed. The results of the test execution are saved to the directory that you specify on the host computer.

Command	Description
<agentImageName>:<imageVersion>	Specifies the name of the agent image and its version to run.
TEST_IMPORT_PATH=<PATH>	Specifies the location of the compressed test assets to be imported into the container. The location path is on the container side and not the host. For example, `/containerTestAssets/ProjectName.zip`. The volume and path names are user-defined and must be consistent.
workbenchimageName:imageVersion	Specifies the name of the workbench image and its version to run.
cmdline	Specifies the existing command-line arguments to define the following details: ◦ Location of the workspace ◦ Project name ◦ Test name ◦ Results file name ◦ Location of the exported logs ◦ Unified report format ◦ Location of the unified report ◦ Unified report file name

Related information

Configuring Docker containers on page 1261

Copying test assets with dependencies

Testing with IBM® Cloud Private

You can automate your testing environment by using Docker container images of the workbench and agents on top of IBM® Cloud Private.

You must use only floating licenses for the product and VT-pack when playing back tests. These licenses should be hosted on a server that can be accessed by the workbench.

Configuring IBM® Cloud Private

To make use of automated workflows in IBM® Cloud Private, you can install it and push the Docker containers to it.

About this task

You must use IBM® Cloud Private 2.1.0.3.

1. Install and set up IBM® Cloud Private-CE.
   For more information, see documentation.
2. Install the IBM® Cloud Private Command Line Interface (CLI).
   For more information, see documentation.
3. Download the Rational® Functional TesterHCL OneTest™ UI and IBM® Rational® Performance TesterHCL OneTest™ Performance Agent container images from the same location you downloaded the product bits. The images are in the form of Passport Advantage (PPA) files. You can load the PPA files directly to IBM® Cloud Private instead of extracting the Docker image out of it.
4. Load the images into Docker repository:

   For Windows, use the command in GitBash:

   ```
   $ docker load -i "imageFileName.tar.gz"
   ```

   For Linux, use the terminal:

   ```
   tar --wildcards --to-command='docker load' -xzf imageName 'images/*'
   ```

   Note: The *imageName* can be the agent image `ibm-rpta-9.2.1.tar.gz` or the workbench image.

   Result

   When the image is loaded, following message is displayed - Loaded image: *imageFileName:versionNumber*
5. Before pushing the Docker image to IBM® Cloud Private cluster, configure the authentication from Docker CLI. For more information, see the documentation.
6. Load the images to the IBM® Cloud Private cluster by using any of the following methods:

   a. Use Docker to log in to the cluster and push the image:
      i. To log in from your client system to the private image registry:

      ```
      $ docker login cluster_CA_domain:8500
      ```

      ii. To tag the images:

      ```
      $ docker tag imagename:tagname cluster_CA_domain:8500/namespacename/imagename:tagname
      ```

      iii. To push the images to the cluster:

      ```
      docker push cluster_CA_domain:8500/namespacename/imagename:tagname
      ```

   b. Use the IBM® Cloud Private CLI to push the images:

   ```
   bx pr load-ppa-archive --archive ibm-rpta-imageVersion.tar.gz
   ```

   ```
   bx pr load-ppa-archive --archive ibm-rtw-imageVersion.tar.gz
   ```

What to do next

You can now set up the playback environment in IBM® Cloud Private. See Running automated tests with containerized agents on IBM Cloud Private on page 1273 and Running automated tests with containerized workbench and agents on IBM Cloud Private on page 1273.

Running automated tests with containerized agents on IBM® Cloud Private

To adopt IBM® Cloud Private fully and manage the entire development to deployment workflow on cloud, you would want to start and stop capabilities with a few clicks. By providing the agents in containers, you can dynamically provision capability as required without procuring the machines and installing the agents.

Before you begin

You must have configured IBM® Cloud Private as per the instructions in Configuring IBM Cloud Private on page 1271.

About this task

You must use only floating licenses to play back the tests.

To automate deployment and playback of the tests, you can choose to create bash scripts for the Kubernetes commands and integrate with Jenkins.

Note: The version number of the container images and the desktop products must match. If you have previous version of the container image, uninstall it and install the current version. To uninstall the image, use these commands:

1. Stop the container by running

   ```
   docker stop "CONTAINER  ID"
   ```

   .
2. Uninstall the image by running the following command:

   ```
   docker rmi -f "image ID"
   ```

What to do next

To stop the agents, in the management console, click **Workloads > Deployments** and under Action select **Remove** to remove each agent deployment.

Running automated tests with containerized workbench and agents on IBM® Cloud Private

To adopt IBM® Cloud Private fully and manage the entire development to deployment workflow on the cloud, you would want to start and stop capabilities with fewer clicks. By providing both the workbench and agents in containers, you can dynamically provision capability as required and run the test automation suites without procuring the machines and installing the products.

Before you begin

You must have configured IBM® Cloud Private as per the instructions in Configuring IBM Cloud Private on page 1271.

About this task

You must use only floating licenses for the product and VT-pack when playing back tests. These licenses should be hosted on a server that can be accessed by the workbench.

Note: The version number of the container images and the desktop products must match. If you have previous version of the container image, uninstall it and install the current version. To uninstall the image, use these commands:

1. Stop the container by running

   ```
   docker stop "CONTAINER  ID"
   ```

   .
2. Uninstall the image by running

   ```
   docker rmi -f "image ID"
   ```

   .

1. In IBM® Cloud Private, create services for the workbench and agents by creating the `services.yml` file. Services are logical set of pods that can provide a single IP address and DNS name by which the pods can be accessed. Creating the services only reserves the IPs and does not create the actual workbench or agent pods. See the sample services.yml file.

   Sample services file:

   ```
   cat services.yml

   apiVersion: v1
   kind: Service
   metadata:
     labels:
       io.kompose.service: <workbench_name>
     name: <workbench_name>
   spec:
     type: NodePort
     ports:
     - name: "7080"
       port: 7080
       targetPort: 7080
     - name: "7443"
       port: 7443
       targetPort: 7443
     selector:
       io.kompose.service: <workbench_name>
   status:
     loadBalancer: {}
   ---
   apiVersion: v1
   kind: Service
   metadata:
   ```

```
  labels:
    io.kompose.service: agent1
  name: agent1
spec:
  ports:
  - name: "7080"
    port: 7080
    targetPort: 7080
  selector:
    io.kompose.service: agent1
status:
  loadBalancer: {}
```

2. Run the command to create the service.

```
kubectl create -f services.yml
```

3. Pass the command to get the IP addresses of the workbench and the agents so that you can use them in the `deployment.yml` file to connect the agents with the workbench.

```
kubectl get services
```

4. Create a `deployment.yml` file to specify the license, agents, workbench, license, and test asset information in the yml file.

Sample deployment file:

```
cat deployment.yml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: <workbench_name>
    pt.classification: workbench
  name: <workbench_name>
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        io.kompose.service: <workbench_name>
        pt.classification: workbench
        pt.name: <workbench_name>
    spec:
      containers:
      - command:
        - cmdline
        - -workspace
        - /runData/<WORKSPACE_NAME>
        - -project
        - <TEST_PROJECT_NAME>
        - -suite
        - Tests/<TEST_SUITE>.testsuite
        - -results
        - autoResults
```



```
        - -stdout
        - -exportlog
        - /runData/<TEST_LOG>.txt


        env:
          - name: RATIONAL_LICENSE_FILE
            value: <licenseServerPort>@<licenseServerIPAddress>
          - name: TEST_IMPORT_PATH
            value: /Tests/<TEST_ASSET_NAME>.zip
        image: mycluster.icp:8500/default/<imageName>:<imageVersion>
        name: <workbench_name>
        ports:
        - containerPort: 7080
        - containerPort: 7443
        resources: {}
      restartPolicy: Always
        volumeMounts:
        - mountPath: /Tests
          name: ft-wb-claim0
#       Optional
#        - mountPath: /runData
#          name: ft-wb-claim1
      restartPolicy: Always
      volumes:
      - name: ft-wb-claim0
        hostPath:
          path: /pathToTestAsset.zip
#      - name: ft-wb-claim1
#        hostPath:
#          path: /pathForWorkspace

status: {}
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: agent1
    pt.classification: agent
  name: agent1
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: agent1
        pt.classification: agent
        pt.name: agent1
    spec:
      containers:
      - env:
        - name: AGENT_NAME
          value: agent1
        - name: AGENT_IP
          value: <ClusterIPAddress>
```

```
        - name: MASTER_NAME
          value: <workbench_name>
        image: mycluster.icp:8500/default/<imageName>:<imageVersion>
        name: agent1
        resources: {}
      restartPolicy: Always
status: {}
```

5. Run the `deployment.yml` file to create the workbench and agent containers.

   ```
   kubectl create -f deployment.yml
   ```

6. Create PersistentVolume and PersistentVolumeClaim in IBM® Cloud Private. To create PersistentVolume, see this topic. To create PersistentVolumeClaim, see this topic.
7. Use IBM® Cloud Private Console to verify that the workbench and agent deployments are created and running successfully. Refer the test execution logs in the workbench and agent pod deployments.
8. Run the commands to stop the workbench and agent containers and verify the test run status in the exported logs.

   ```
   kubectl delete -f deployment.yml
   kubectl delete -f services.yml
   ```

Related information

Running automated tests with containerized agents on IBM Cloud Private on page 1273

Configuration of mobile test runs

After you record the tests for mobile applications by using Rational® Functional TesterHCL OneTest™ UI, you can play back the recorded tests by using Rational® Functional TesterHCL OneTest™ UI. The test reports help you to analyze the functional and performance issues in the mobile applications developed.

Rational® Functional TesterHCL OneTest™ UI supports the testing of mobile applications developed for the following types of mobile operating systems:

- The Android mobile operating system.
- The iOS mobile operating system.

Running mobile tests for Android mobile applications

Before you can use Rational® Functional TesterHCL OneTest™ UI to play back the recorded tests, you must record the tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Recording mobile tests for Android applications on page 692.

By using Rational® Functional TesterHCL OneTest™ UI, you can play back the recorded mobile test on Android devices or emulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

You can find information about the tasks that you can perform when you want to play back the recorded tests for Android mobile applications as follows:

- Running mobile tests on an Android device or emulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1279
- Running mobile tests on an Android device or emulator connected to a remote agent computer on page 1281
- Running mobile tests on Android devices on BitBar Cloud on page 1284
- Running mobile tests on Android devices on BrowserStack Cloud on page 1286
- Running mobile tests on Android devices on Perfecto Cloud on page 1290
- Playing back mobile tests on Android devices on pCloudy Cloud on page 1288
- Running mobile tests as an AFT suite on page 1339
- Running mobile tests for Android applications from the command-line on page 1415

Running mobile tests for iOS mobile applications

Before you can play back the recorded tests, you must record the tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Recording mobile tests for iOS applications on page 701.

You can play back the recorded mobile test on iOS devices or simulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

Note: You cannot connect simulators to device clouds.

You can find information about the tasks that you can perform when you want to play back the recorded tests for iOS mobile applications as follows:

- Running mobile tests on an iOS device or simulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1311
- Running mobile tests on an iOS device or simulator connected to a remote computer on page 1313

- Running mobile tests on iOS devices on BitBar Cloud on page 1315
- Running mobile tests on iOS devices on BrowserStack Cloud on page 1317
- Running mobile tests on iOS devices on Perfecto Cloud on page 1321
- Running mobile tests on iOS devices on pCloudy Cloud on page 1319
- Running mobile tests as an AFT suite on page 1339
- Running mobile tests for iOS applications from the command-line on page 1424

Running mobile tests for Android mobile applications

After you record the tests for Android mobile applications by using Rational® Functional TesterHCL OneTest™ UI, you can play back the recorded tests by using Rational® Functional TesterHCL OneTest™ UI.

Before you can use Rational® Functional TesterHCL OneTest™ UI to play back the recorded tests, you must record the tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Recording mobile tests for Android applications on page 692.

By using Rational® Functional TesterHCL OneTest™ UI, you can play back the recorded mobile test on Android devices or emulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

You can find information about the tasks that you can perform when you want to play back the recorded tests for Android mobile applications as follows:

- Running mobile tests on an Android device or emulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1279
- Running mobile tests on an Android device or emulator connected to a remote agent computer on page 1281
- Running mobile tests on Android devices on BitBar Cloud on page 1284
- Running mobile tests on Android devices on BrowserStack Cloud on page 1286
- Running mobile tests on Android devices on Perfecto Cloud on page 1290
- Playing back mobile tests on Android devices on pCloudy Cloud on page 1288
- Running mobile tests as an AFT suite on page 1339
- Running mobile tests for Android applications from the command-line on page 1415

Running mobile tests on an Android device or emulator connected to a computer that runs Rational® Functional TesterHCL OneTest™ UI

You can run mobile tests on Android mobile devices or emulators that are connected to a computer that runs Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:
  - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
  - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.
- Installed the Android application that you want to test on the Android device or emulator.
- Connected and started the Android device or emulator to the computer that you want to use for testing Android mobile applications. You can connect the Android device to the computer through a USB cable.
- Ensured that the UI Test Agent is running on the computer that runs Rational® Functional TesterHCL OneTest™ UI.



Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.

3. Click the **Run Test**  icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device or emulator to play back the test for each mobile test.

4. Select the mobile device or emulator to play back the test from the **Run using** list and click **Next**.

   The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

5. Click **Next**.

   The **Performance Measurement** dialog is displayed.

6. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

   Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

7. Click **Finish**.

Results

The test runs on the selected mobile device or emulator. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on an Android device or emulator connected to a remote agent computer

You can run mobile tests on an Android device or emulator that is connected to a remote agent computer.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:

  - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
  - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.
- Installed the Android application that you want to test on the Android device or emulator.
- Connected and started the Android device or emulator to the computer that you want to use for testing Android mobile applications. You can connect the Android device to the computer through a USB cable.
- Ensured that the UI Test Agent is running on the remote agent computer.



Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Click the **Mobile Device** tab, and then perform the following steps:

   a. Select the **Host** check box in the **UI Test Agent** panel, and then enter the IP address of the remote computer in the **Host** field.

   Note: The port in use by the UI Test Agent on the computer is automatically populated in the **Port** field.

   b. Select the **Android Device** check box in the **Android Device** panel, and then enter the name of the Android device or emulator that is connected to the computer in the **Android Device** field.

   The **Is real device** check box is enabled.

   c. Perform any of following actions:

- If you are using an Android device, select the **Is real device** check box.
- If you are using an emulator, retain the **Is real device** check box as deselected or clear the selection, if selected.

d. Click **Apply and Close**.

5. Click the **Run Test**  icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device or emulator to play back the test for each mobile test.

6. Select the mobile device or emulator to play back the test from the **Run using** list and click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: There is no action to be performed by you for mobile tests.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

The test runs on the Android device or emulator that is connected to the remote computer. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on Android devices on BitBar Cloud

You can run mobile tests on an Android device that is connected to the BitBar Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application that you want to test on an Android device in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:
   a. Click the **Mobile Device Cloud** tab.
   b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.
   c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.

Option	Action
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  . **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Test**  icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

The test runs on the Android device that is connected to the BitBar Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on Android devices on BrowserStack Cloud

You can run mobile tests on the BrowserStack Cloud so that you can choose devices from a wide range of Android devices that are available for testing.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.apk` file of the Android application to the BrowserStack server for testing. For more information, refer to Live Testing in the BrowserStack documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

   d. Click **Apply and Close**.

   Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.
5. Click the **Run Test** icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.
6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

   Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

You have run the mobile test on multiple Android devices that are avialble on the BrowserStack Cloud.

What to do next

You can view the test result that is displayed as a unified report for mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Playing back mobile tests on Android devices on pCloudy Cloud

You can run mobile tests on an Android device that is connected to the pCloudy Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application that you want to test on an Android device in the pCloudy cloud. For more information, refer to Upload Android App in the pCloudy documentation.

- Installed the Android application on an Android device in the pCloudy cloud. For more information, refer to Installing Android app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:
   a. Click the **Mobile Device Cloud** tab.
   b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.
   c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

   d. Click **Apply and Close**.

   Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.
5. Click the **Run Test** icon.

The **Run Configuration** dialog is displayed.

Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

The test runs on the Android device that is connected to the pCloudy Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on Android devices on Perfecto Cloud

You can run mobile tests on an Android device that is connected to the Perfecto Mobile Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.
- Installed the `.apk` file of the Android application that you want to test on an Android device in the Perfecto Mobile Cloud. For information, refer to Manage Apps in the Perfecto documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:
   a. Click the **Mobile Device Cloud** tab.
   b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.
   c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

   d. Click **Apply and Close**.

   Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.
5. Click the **Run Test** icon.

The **Run Configuration** dialog is displayed.

Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

The test runs on the Android device that is connected to the Perfecto Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests for Android mobile applications

After you create the compound tests for Android mobile applications by using Rational® Functional TesterHCL OneTest™ UI, you can play back the compound tests by using Rational® Functional TesterHCL OneTest™ UI.

Before you can use Rational® Functional TesterHCL OneTest™ UI to play back the recorded tests, you must create the compound tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Creating a compound test on page 738.

By using Rational® Functional TesterHCL OneTest™ UI, you can play back the compound test on Android devices or emulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

You can find information about the tasks that you can perform when you want to play back the recorded tests for Android mobile applications as follows:

- Running compound tests on Android devices or emulators connected to a computer that runs IBM Rational Functional TesterHCL OneTest UI on page 1293.
- Running compound tests on Android devices or emulators connected to a remote agent computer on page 1296.
- Running compound tests on Android devices on BitBar Cloud on page 1298.
- Running compound tests on Android devices on BrowserStack Cloud on page 1302.
- Running compound tests on Android devices on pCloudy Cloud on page 1304.
- Running compound tests on Android devices on Perfecto Cloud on page 1307.

Running compound tests on Android devices or emulators connected to a computer that runs IBM® Rational® Functional TesterHCL OneTest™ UI

You can run compound tests that contain mobile tests on Android mobile devices or emulators that are connected to a computer that runs Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:

  - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
  - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.
- Installed the Android application that you want to test on the Android device or emulator. You can connect the Android device to the computer through a USB cable.
- Connected and started the Android device or emulator to the computer that you want to use for testing Android mobile applications.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click the **Run Compound Test**  icon.

   The **Run Configuration** dialog is displayed.

   Perform any of the following actions:
   - To run mobile tests that are in the compound test on a specific device or emulator, go to step 4 on page 1294.
   - To simultaneously run the mobile tests that are in the compound test on multiple devices or emulators, go to step 5 on page 1295.
4. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected.

   b. Select a device or emulator from the list.

   c. Select the device or emulator for each of the tests by repeating the action in Step 4.a on page 1294 and Step 4.b on page 1294.

   d. Click **Next.**

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 6 on page 1295.

5. Select the **Run on several devices and browsers in parallel** option, if it is not already selected.

   Mobile devices and emulators that are connected to the computer are displayed.

   You must perform the following steps:

   a. Select the devices on which you want to run the compound test.

   b. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

      Note: There is no action to be performed by you for mobile tests.

   c. Go to step 6 on page 1295.

6. Click **Next**.

   The **Performance Measurement** dialog is displayed.

7. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

8. Click **Finish**.

Results

The test runs on the selected mobile devices or emulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on Android devices or emulators connected to a remote agent computer

You can run compound tests that contain mobile tests on Android mobile devices or emulators that are connected to a remote agent computer.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Installed the Android SDK on the computer that you want to use for testing Android mobile applications.
- Set or changed the value of the **ANDROID_HOME** environment variable on the computer that you want to use for testing Android mobile applications for the following operating systems:
  - Windows operating systems, see Setting or changing the ANDROID_HOME path in Windows operating systems on page 1241.
  - Linux operating systems, see Setting or changing the ANDROID_HOME path in Linux operating systems on page 1239.
  - Mac operating systems, see Setting or changing the ANDROID_HOME path in Mac operating systems on page 1240.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Click the **Mobile Device** tab, and then perform the following steps:

a. Select the **Host** check box in the **UI Test Agent** panel, and then enter the IP address of the remote computer in the **Host** field.

Note: The port in use by the UI Test Agent on the computer is automatically populated in the **Port** field.

b. Select the **Android Device** check box in the **Android Device** panel, and then enter the name of the Android device or emulator that is connected to the computer in the **Android Device** field.

The **Is real device** check box is enabled.

c. Perform any of following actions:

- If you are using an Android device, select the **Is real device** check box.
- If you are using an emulator, retain the **Is real device** check box as deselected or clear the selection, if selected.

d. Click **Apply and Close**.

5. Click the **Run Compound Test** icon.

The **Run Configuration** dialog box is displayed.

Perform any of the following actions:

- To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1297.
- To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1297.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the remote agent computer.

b. Select a device from the list.

c. Repeat steps 6.a on page 1297 and 6.b on page 1297, for each of the tests in the Compound test.

d. Click **Next.**

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 8 on page 1298.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.

All the mobile devices and emulators that are connected to the remote agent computer are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Go to step 8 on page 1298.

8. Click **Next**.

The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

The test runs on the selected mobile devices or emulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on Android devices on BitBar Cloud

You can run compound tests that contain mobile tests on Android mobile devices that are connected to the BitBar Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application that you want to test on an Android device in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .

Option	Action
	Note: Clicking the **Refresh projects and device groups** enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1300.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1301.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the BitBar Cloud.

   b. Select a device from the list.

   c. Repeat steps 6.a on page 1300 and 6.b on page 1300, for each of the tests in the Compound test.

   d. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 8 on page 1301.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
   All the mobile devices and emulators that are connected to the BitBar Cloud are displayed.

   You must perform the following steps:

   a. Select the devices on which you want to run the test.

   b. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

      Note: There is no action to be performed by you for mobile tests.

   c. Go to step 8 on page 1301.

8. Click **Next**.

   The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

The test runs on the selected mobile devices or emulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on Android devices on BrowserStack Cloud

You can run compound tests for mobile tests on the BrowserStack Cloud so that you can choose devices from a wide range of Android devices that are available for testing.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.apk` file of the Android application to the BrowserStack server for testing. For more information, refer to Live Testing in the BrowserStack documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:
   a. Click the **Mobile Device Cloud** tab.
   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.
   c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.

Option	Action
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

The **Run Configuration** dialog is displayed.

Perform any of the following actions:

- To run each mobile test that is in the compound test on a specific device, go to step 6 on page 1303.
- To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1303.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the BrowserStack Cloud.

b. Select a device from the list.

c. Repeat steps 6.a on page 1303 and 6.b on page 1303, for each of the tests in the Compound test.

d. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 8 on page 1304.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
All the mobile devices and emulators that are connected to the BrowserStack Cloud are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Go to step 8 on page 1304.

8. Click **Next**.

The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

You have run the compound test on multiple Android devices that are avialble on the BrowserStack Cloud.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on Android devices on pCloudy Cloud

You can run compound tests that contain mobile tests on Android mobile devices that are connected to the pCloudy Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application that you want to test on an Android device in the pCloudy cloud. For more information, refer to Upload Android App in the pCloudy documentation.
- Installed the Android application on an Android device in the pCloudy cloud. For more information, refer to Installing Android app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.

Option	Action
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

d. Click **Apply and Close**.

Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test**  icon.

   The **Run Configuration** dialog box is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1306.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1306.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the pCloudy Cloud.

   b. Select a device from the list.

   c. Repeat steps 6.a on page 1306 and 6.b on page 1306, for each of the tests in the Compound test.

   d. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

      Note: There is no action to be performed by you for mobile tests.

   e. Go to step 8 on page 1307.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
   All the mobile devices and emulators that are connected to the pCloudy Cloud are displayed.

   You must perform the following steps:

   a. Select the devices on which you want to run the test.

   b. Click **Next**.

The **Advanced Playback Options** dialog box is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Go to step 8 on page 1307.

8. Click **Next**.

   The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

   Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

The test runs on the selected mobile devices or emulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on Android devices on Perfecto Cloud

You can run compound tests that contain mobile tests on Android mobile devices that are connected to the Perfecto Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for Android applications on page 692.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.

- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.
- Installed the `.apk` file of the Android application that you want to test on an Android device in the Perfecto Mobile Cloud. For information, refer to Manage Apps in the Perfecto documentation.
- Installed the Android application that you want to test on the Android device.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

d. Click **Apply and Close**.

Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog box is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1309.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1309.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the Perfecto Cloud.

   b. Select a device from the list.

   c. Repeat steps 6.a on page 1309 and 6.b on page 1309, for each of the tests in the Compound test.

   d. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

      Note: There is no action to be performed by you for mobile tests.

   e. Go to step 8 on page 1310.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
   All the mobile devices and emulators that are connected to the Perfecto Cloud are displayed.

   You must perform the following steps:

   a. Select the devices on which you want to run the test.

   b. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

      Note: There is no action to be performed by you for mobile tests.

   c. Go to step 8 on page 1310.

8. Click **Next**.

   The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

   Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests for iOS mobile applications

You can run the tests for iOS mobile applications that are recorded by using Rational® Functional TesterHCL OneTest™ UI.

Before you can play back the tests, you must have recorded the tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Recording mobile tests for iOS applications on page 701.

Before you can play back the recorded mobile test, you must complete the following tasks:

- Installed the Xcode and the CLI tools for Xcode on the computer that you want to use for testing iOS mobile applications.
- Connected and started the iOS device that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.
- Installed the iOS application that you want to test on the iOS device.

You can play back the recorded mobile test on iOS devices or simulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.

- Perfecto Mobile Cloud.
- pCloudy Cloud.

Note: You cannot connect simulators to device clouds.

To play back a mobile test, multiple mobile tests, a compound test, or an AFT Suite, after you select the test, you can select the location where the iOS devices or simulators are connected, and then specify the iOS device or simulator on which you want to run the test.

When you play back the Compound Test containing mobile tests, you can specify different iOS devices or simulators for each of the mobile test in the Compound Test.

See Configuration of mobile test runs on page 1277.

You can find information about the tasks that you can perform when you want to play back the recorded tests for iOS mobile applications as follows:

- Running mobile tests on an iOS device or simulator connected to a computer that runs Rational Functional TesterHCL OneTest UI on page 1311
- Running mobile tests on an iOS device or simulator connected to a remote computer on page 1313
- Running mobile tests on iOS devices on BitBar Cloud on page 1315
- Running mobile tests on iOS devices on BrowserStack Cloud on page 1317
- Running mobile tests on iOS devices on Perfecto Cloud on page 1321
- Running mobile tests on iOS devices on pCloudy Cloud on page 1319
- Running mobile tests as an AFT suite on page 1339
- Running mobile tests for iOS applications from the command-line on page 1424

Running mobile tests on an iOS device or simulator connected to a computer that runs Rational® Functional TesterHCL OneTest™ UI

You can run mobile tests on iOS mobile devices or simulators that are connected to a computer that runs Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Installed the Xcode and the CLI tools for Xcode on the computer that you want to use for testing iOS mobile applications.
- Installed the iOS mobile application that you want to test on the iOS device or simulator.
- Connected and started the iOS device that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.

- Ensured that the UI Test Agent is running on the computer that runs Rational® Functional TesterHCL OneTest™ UI.

  Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.

3. Click the **Run Test**  icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device or simulator to play back the test for each mobile test.

4. Select the mobile device or simulator to play back the test from the **Run using** list and click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: There is no action to be performed by you for mobile tests.

5. Click **Next**.

   The **Performance Measurement** dialog is displayed.

6. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

7. Click **Finish**.

Results

The test runs on the selected mobile device or simulator. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on an iOS device or simulator connected to a remote computer

You can run mobile tests on an iOS device or simulator that is connected to a remote agent computer.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Installed the Xcode and the CLI tools for Xcode on the computer that you want to use for testing iOS mobile applications.
- Installed the iOS mobile application that you want to test on the iOS device or simulator.
- Connected and started the iOS device that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.
- Ensured that the UI Test Agent is running on the remote agent computer.

  Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

- Ensured that you have the following information if you want to run the test on a real device:
  - Apple Team ID
  - Version of the platform

- Port

  Note: This is required only when the port is not the default port.

- Name of the iOS mobile device

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Click the **Mobile Device** tab, and then perform the following steps:
   a. Select the **Host** check box in the **UI Test Agent** panel, and then enter the IP address of the remote computer in the **Host** field.

      Note: The port in use by the UI Test Agent on the computer is automatically populated in the **Port** field.

   b. Select the **iOS Device** checkbox in the **iOS Device** panel, and then enter the name of the iOS device or simulator that is connected to the computer in the **iOS Device** field.

      The **Platform Version** field is enabled.

   c. Enter the version of the platform in the **Platform Version** field.

   d. Select the **Apple Team ID** checkbox in the **iOS Device** panel, and then enter the name of the Apple Developer Team ID provided by Apple in the **Apple Team ID** field.

   e. Enter the role that you are assigned by the Apple Developer Team in the **Role** field.

   f. Click **Apply and Close**.
5. Click the **Run Test** icon.

The **Run Configuration** dialog is displayed.

Note: If you are using a compound test that contains mobile tests, you can specify the mobile device or simulator to play back the test for each mobile test.

6. Select the mobile device or simulator to play back the test from the **Run using** list and click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: There is no action to be performed by you for mobile tests.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

8. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

   a. Select the **Resource Monitoring** option.

   b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

      Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

9. Click **Finish**.

Results

The test runs on the iOS device or simulator that is connected to the remote computer. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running mobile tests on iOS devices on BitBar Cloud

You can run mobile tests on an iOS device that is connected to the BitBar Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Installed the `.ipa` or `.app` file of the iOS application that you want to test in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .  **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Test**  icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

   Note: For mobile tests, you can skip this step.

8. Click **Finish**.

Results

The test runs on the iOS device that is connected to the BitBar Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests on iOS devices on BrowserStack Cloud

You can run mobile tests on the BrowserStack Cloud so that you can choose devices from a wide range of iOS devices that are available for testing.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.ipa` or `.app` file of the iOS application to the BrowserStack server for testing. For more information, refer to Live Testing in the BrowserStack documentation.
- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

   d. Click **Apply and Close**.

   Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Test** icon.

The **Run Configuration** dialog is displayed.

Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

   Note: For mobile tests, you can skip this step.

8. Click **Finish**.

Results

You have run the mobile test on multiple iOS devices that are avialble on the BrowserStack Cloud.

What to do next

You can view the test result that is displayed as a unified report for mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests on iOS devices on pCloudy Cloud

You can run mobile tests on an iOS device that is connected to the pCloudy Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.ipa` file of the iOS application that you want to test on an iOS device in the pCloudy Cloud. For more information, refer to Upload iOS App in the pCloudy documentation.
- Installed the iOS application on an iOS device in the pCloudy Cloud. For more information, refer to Installing iOS app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

   d. Click **Apply and Close**.

   Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Test** icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

   Note: For mobile tests, you can skip this step.

8. Click **Finish**.

Results

The test runs on the iOS device that is connected to the pCloudy Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests on iOS devices on Perfecto Cloud

You can run mobile tests on an iOS device that is connected to the Perfecto Mobile Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.
- Installed the `.ipa` file of the iOS application that you want to test on an iOS device in the Perfecto Mobile Cloud. For information, refer to Manage Apps in the Perfecto documentation.
- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the mobile test from the **Test Navigator** pane.

   The test window is displayed.

3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.

4. Perform the following steps in the **UI Test Playback** pane:

a. Click the **Mobile Device Cloud** tab.

b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

d. Click **Apply and Close**.

Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Test** icon.

   The **Run Configuration** dialog is displayed.

   Note: If you are using a compound test that contains mobile tests, you can specify the mobile device to play back the test for each mobile test.

6. Select the mobile device to play back the test from the **Run using** list, and then click **Next**.

   The **Advanced Playback Options** dialog is displayed.

   Note: For mobile tests, you can skip this step.

7. Click **Next**.

   The **Performance Measurement** dialog is displayed.

   Note: For mobile tests, you can skip this step.

8. Click **Finish**.

Results

The test runs on the iOS device that is connected to the Perfecto Cloud. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running compound tests for iOS mobile applications

After you create the compound tests for iOS mobile applications by using IBM® Rational® Functional TesterHCL OneTest™ UI, you can play back the compound tests.

Before you can play back a compound test, you must create compound tests for the mobile applications by using Rational® Functional TesterHCL OneTest™ UI. See Creating a compound test on page 738.

You can play back the compound test on iOS devices or simulators that are connected to any of the following computers or device clouds:

- Computer that runs Rational® Functional TesterHCL OneTest™ UI.
- Remote agent computer.
- BitBar Cloud.
- BrowserStack Cloud.
- Perfecto Mobile Cloud.
- pCloudy Cloud.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

You can find information about the tasks that you can perform when you want to play back the recorded tests for iOS mobile applications as follows:

- Running compound tests on iOS devices or simulators connected to a computer that runs IBM Rational Functional TesterHCL OneTest UI on page 1323
- Running compound tests on iOS devices or simulators connected to a remote agent computer on page 1326
- Running compound tests on iOS devices on BitBar Cloud on page 1329
- Running compound tests on iOS devices on BrowserStack Cloud on page 1331
- Running compound tests on iOS devices on pCloudy Cloud on page 1334
- Running compound tests on iOS devices on Perfecto Cloud on page 1336

Running compound tests on iOS devices or simulators connected to a computer that runs IBM® Rational® Functional TesterHCL OneTest™ UI

After you create the compound tests for iOS mobile applications by using Rational® Functional TesterHCL OneTest™ UI, you can play back the compound tests by using Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Installed the Xcode and the CLI tools for Xcode on the computer that you want to use for testing iOS mobile applications.
- Connected and started the iOS device or simulator that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.
- Ensured that the UI Test Agent is running on the computer that runs Rational® Functional TesterHCL OneTest™ UI.

Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 4 on page 1324.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 5 on page 1325.
4. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:
   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected.
   b. Select a device from the list.

c. Repeat steps 4.a on page 1324 and 4.b on page 1324, for each of the tests in the Compound test.
d. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 6 on page 1325.

5. Select the **Run on several devices and browsers in parallel** option, if not selected.
All the mobile devices and emulators that are connected to the computer are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.
b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Go to step 6 on page 1325.

6. Click **Next**.

The **Performance Measurement** dialog is displayed.

7. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

8. Click **Finish**.

Results

The test runs on the selected mobile devices or simulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on iOS devices or simulators connected to a remote agent computer

You can run compound tests that contain mobile tests on iOS mobile devices or simulators that are connected to a remote agent computer.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Installed the Xcode and the CLI tools for Xcode on the computer that you want to use for testing iOS mobile applications.
- Installed the iOS mobile application that you want to test on the iOS device or simulator.
- Connected and started the iOS device that you want to use for testing iOS mobile applications. You can connect the iOS device to the computer through a USB cable.
- Ensured that the UI Test Agent is running on the remote agent computer.

  Tip: You can hover the cursor over the **System Tray** to verify that the UI Test Agent is running and also view the port it uses.

- Ensured that you have the following information if you want to run the test on a real device:
  - Apple Team ID
  - Version of the platform
  - Port

    Note: This is required only when the port is not the default port.

  - Name of the iOS mobile device

About this task

When you run mobile tests, you can enable Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile applications or devices. You can select the Resource Monitoring option and set the time intervals for the collection of the metrics during the test run.

Important: The collection of Resource Monitoring metrics and display of the data graphically is supported only when you run a compound test on a single device.

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Click the **Mobile Device** tab, and then perform the following steps:

   a. Select the **Host** check box in the **UI Test Agent** panel, and then enter the IP address of the remote computer in the **Host** field.

      Note: The port in use by the UI Test Agent on the computer is automatically populated in the **Port** field.

   b. Select the **iOS Device** checkbox in the **iOS Device** panel, and then enter the name of the iOS device or simulator that is connected to the computer in the **iOS Device** field.

      The **Platform Version** field is enabled.

   c. Enter the version of the platform in the **Platform Version** field.

   d. Select the **Apple Team ID** checkbox in the **iOS Device** panel, and then enter the name of the Apple Developer Team ID provided by Apple in the **Apple Team ID** field.

   e. Enter the role that you are assigned by the Apple Developer Team in the **Role** field.

   f. Click **Apply and Close**.

5. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog box is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1327.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1328.
6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected.

b. Select a device from the list.

c. Repeat steps 6.a on page 1328 and 6.b on page 1328, for each of the tests in the Compound test.

d. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Go to step 8 on page 1328.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Go to step 8 on page 1328.

8. Click **Next**.

The **Performance Measurement** dialog is displayed.

9. Perform the following steps if you want Rational® Functional TesterHCL OneTest™ UI to collect the Resource Monitoring metrics during the test run:

a. Select the **Resource Monitoring** option.

b. Set the interval to collect the metrics by entering a number in the **Polling interval** field.

Note: The minimum value that you can enter in the **Polling interval** field is 1000 milliseconds.

10. Click **Finish**.

Results

The test runs on the selected mobile devices or simulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

If you opted to collect the Resource Monitoring metrics, you can view the metrics displayed graphically in the UI Test Statistical report. You can analyze the Resource Monitoring metrics to manage the resources of the device or the mobile application under test. See UI Test Statistical report on page 1469.

Running compound tests on iOS devices on BitBar Cloud

You can run compound tests that contain mobile tests on iOS mobile devices that are connected to the BitBar Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Installed the `.ipa` or `.app` file of the iOS application that you want to test in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .

Option	Action
	Note: Clicking the **Refresh projects and device groups** enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog is displayed.

   Perform any of the following actions:
   - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1330.
   - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1331.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

   a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the BitBar Cloud.

   b. Select a device from the list.

   c. Repeat steps 6.a on page 1330 and 6.b on page 1330, for each of the tests in the Compound test.

   d. Click **Next**.

      The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

f. Click **Finish**.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
All the mobile devices and emulators that are connected to the BitBar Cloud are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

d. Click **Finish**.

Results

The test runs on the selected mobile devices or simulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running compound tests on iOS devices on BrowserStack Cloud

You can run compound tests for mobile tests on the BrowserStack Cloud so that you can choose devices from a wide range of iOS devices that are available for testing.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.ipa` or `.app` file of the iOS application to the BrowserStack server for testing. For more information, refer to Live Testing in the BrowserStack documentation.
- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

The **Run Configuration** dialog is displayed.

Perform any of the following actions:

- To run each mobile test that is in the compound test on a specific device, go to step 6 on page 1333.
- To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1333.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the BrowserStack Cloud.

b. Select a device from the list.

c. Repeat steps 6.a on page 1333 and 6.b on page 1333, for each of the tests in the Compound test.

d. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: For mobile tests, you can skip this step.

e. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: For mobile tests, you can skip this step.

f. Click **Finish**.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
All the mobile devices and emulators that are connected to the BrowserStack Cloud are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

d. Click **Finish**.

Results

You have run the compound test on multiple iOS devices that are avialble on the BrowserStack Cloud.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running compound tests on iOS devices on pCloudy Cloud

You can run compound tests that contain mobile tests on iOS mobile devices that are connected to the pCloudy Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.ipa` file of the iOS application that you want to test on an iOS device in the pCloudy Cloud. For more information, refer to Upload iOS App in the pCloudy documentation.
- Installed the iOS application on an iOS device in the pCloudy Cloud. For more information, refer to Installing iOS app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.

4. Perform the following steps in the **UI Test Playback** pane:

    a. Click the **Mobile Device Cloud** tab.

    b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

    c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

    d. Click **Apply and Close**.

    Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

    The **Run Configuration** dialog is displayed.

    Perform any of the following actions:
    - To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1335.
    - To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1336.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

    a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the pCloudy Cloud.

    b. Select a device from the list.

    c. Repeat steps 6.a on page 1335 and 6.b on page 1335, for each of the tests in the Compound test.

    d. Click **Next**.

    The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

e. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

f. Click **Finish**.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
All the mobile devices and emulators that are connected to the pCloudy Cloud are displayed.

You must perform the following steps:

a. Select the devices on which you want to run the test.

b. Click **Next**.

The **Advanced Playback Options** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

c. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

d. Click **Finish**.

Results

The test runs on the selected mobile devices or simulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running compound tests on iOS devices on Perfecto Cloud

You can run compound tests that contain mobile tests on iOS mobile devices that are connected to the Perfecto Cloud.

Before you begin

You must have completed the following tasks:

- Recorded mobile tests. See Recording mobile tests for iOS applications on page 701.
- Created a compound test that contains mobile tests, see Creating a compound test on page 738.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.
- Installed the `.ipa` file of the iOS application that you want to test on an iOS device in the Perfecto Mobile Cloud. For information, refer to Manage Apps in the Perfecto documentation.

- Installed the iOS application that you want to test on the iOS device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the compound test that contains the mobile tests from the **Test Navigator** pane.
   The test window is displayed.
3. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
4. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

   d. Click **Apply and Close**.

   Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

5. Click the **Run Compound Test** icon.

   The **Run Configuration** dialog box is displayed.

Perform any of the following actions:

- To run each mobile test that are in the compound test on a specific device, go to step 6 on page 1338.
- To simultaneously run all the tests that are in the compound test on the same set of devices, go to step 7 on page 1338.

6. Clear the **Run on several devices and browsers in parallel** option, if selected, and then perform the following actions:

    a. Click the **Run using** option for each test, to view the mobile devices or emulators that are connected to the Perfecto Cloud.

    b. Select a device from the list.

    c. Repeat steps 6.a on page 1335 and 6.b on page 1335, for each of the tests in the Compound test.

    d. Click **Next**.

    The **Advanced Playback Options** dialog is displayed.

    Note: There is no action to be performed by you for mobile tests.

    e. Click **Next**.

    The **Performance Measurement** dialog is displayed.

    Note: There is no action to be performed by you for mobile tests.

    f. Click **Finish**.

7. Select the **Run on several devices and browsers in parallel** option, if not selected.
   All the mobile devices and emulators that are connected to the Perfecto Cloud are displayed.

   You must perform the following steps:

    a. Select the devices on which you want to run the test.

    b. Click **Next**.

    The **Advanced Playback Options** dialog is displayed.

    Note: There is no action to be performed by you for mobile tests.

    c. Click **Next**.

The **Performance Measurement** dialog is displayed.

Note: There is no action to be performed by you for mobile tests.

d. Click **Finish**.

Results

The test runs on the selected mobile devices or simulators. The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Configuration of AFT Suite runs for mobile tests

When you create mobile tests and want to use the distributed effort feature inIBM® Rational® Functional TesterHCL OneTest™ UI, you must configure the AFT Suite run for the recorded mobile tests before you can view the test results.

Before you configure a run of an AFT Suite, you must create the AFT Suite as an AFT XML. See Accelerated Functional Tests on page 745.

You can find the following information about configuring different test runs as an AFT Suite:

- Running mobile tests as an AFT suite on page 1339
- Running mobile tests as an AFT Suite on BitBar Cloud on page 1340
- Running mobile tests as an AFT Suite on BrowserStack Cloud on page 1342
- Running mobile tests as an AFT Suite on pCloudy Cloud on page 1344
- Running mobile tests as an AFT Suite on Perfecto Cloud on page 1345
- Running mobile tests as an AFT Suite on multiple mobile clouds on page 1347

Running mobile tests as an AFT suite

After you create an Accelerated Functional Test (AFT) suite for mobile tests, you run the AFT suite by using the **Run Distributed Tests** option.

Before you begin

You must have completed the following tasks:

- Created an AFT suite with mobile tests. See Creating an AFT suite for mobile tests on page 749.
- The configured UI Test Agent must be running.

1. Right-click the XML file in the **Test Navigator** pane, and then click **Run Distributed Tests**.

   The **Run Accelerated Functional Test** dialog box is displayed.

   The following options are displayed in the **Run Accelerated Functional Test** dialog box:
   - **Re-run failed tests only from the last playback**: Select this checkbox if you want to rerun the failed tests only from the last playback. The failed test is again run on the same device and location on which it previously failed.

     Note: This option is enabled only if the test was already run at least once.

   - **Fix the browser-driver incompatibility**: This option is not applicable for mobile tests and it is disabled.
2. Click **OK**.

Results

You have run mobile tests as an AFT suite.

Related information

Running mobile tests for Android applications from the command-line on page 1415

Running mobile tests for iOS applications from the command-line on page 1424

Running mobile tests as an AFT Suite on BitBar Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to BitBar Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the BitBar cloud on page 754.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Uploaded the `.ipa` or `.app` file of the iOS application or the `.apk` file of the Android application that you want to test in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the Android and iOS applications that you want to test on the device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

a. Click the **Mobile Device Cloud** tab.

b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.

c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .  **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

 **Note:** When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on BrowserStack Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on multiple devices connected to the BrowserStack Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on BrowserStack Cloud on page 757.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.ipa` or `.app` file of the iOS application or the `.apk` file of the Android application to the BrowserStack server for testing. For information, refer to Live Testing in the BrowserStack documentation.
- Installed the Android and iOS applications that you want to test on the device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.

c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

What to do next

You can view the test result that is displayed as a unified report for mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on pCloudy Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to pCloudy Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the pCloudy cloud on page 760.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application or the `.ipa` file of the iOS application that you want to test in the pCloudy Cloud. For more information, refer to Upload Android App or Upload iOS App in the pCloudy documentation.
- Installed the Android application or iOS application that you want to test on the device. For more information, refer to Installing Android app or Installing iOS app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

d. Click **Apply and Close**.

Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on Perfecto Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to Perfecto Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the Perfecto cloud on page 763.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.

- Uploaded the `.ipa` file of the iOS application or the `.apk` file of the Android application that you want to test in the Perfecto Cloud. For information, refer to Manage Apps in the Perfecto documentation.
- Installed the Android and iOS applications that you want to test on the device.

About this task

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

   d. Click **Apply and Close**.

   Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on multiple mobile clouds

After you create an AFT Suite for mobile tests, you can run the tests simultaneously on devices connected to multiple mobile clouds to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT suite with mobile tests, See Creating an AFT Suite to run the tests on multiple mobile clouds on page 767.
- You must have set up accounts and obtained valid credentials for more than one mobile cloud as mentioned below:
  - BitBar Cloud: The host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
  - pCloudy Cloud: The host name or the URL of the pCloudy Cloud instance, the user name, and an API key to authenticate the connection.
  - Perfecto Cloud: The host name or the URL of the Perfecto Cloud instance, and the security token to authenticate the connection.
- Uploaded the `.ipa` file of the iOS application or the `.apk` file of the Android application that you want to test in the mobile cloud. For more information, refer to the following:
  - BitBar: Live Testing in the BitBar documentation.
  - pCloudy: Upload Android App or Upload iOS App in the pCloudy documentation.
  - Perfecto: Manage Apps in the Perfecto documentation.
- Installed the Android or iOS application that you want to test on the device.

Note: To install the application in a device on the pCloudy Cloud, refer to Installing Android app or Installing iOS app in the pCloudy documentation.

- If the test to be run on pCloudy Cloud, verified that the device you want to test is available and in the Released state on the pCloudy cloud.

About this task

To run the AFT Suite on multiple clouds, you must configure the mobile clouds with IBM® Rational® Functional TesterHCL OneTest™ UI. You need to configure only those mobile clouds, to which the devices you want to run the tests are connected.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Click the **Mobile Device Cloud** tab, and perform the following steps to configure the mobile clouds with Rational® Functional TesterHCL OneTest™ UI:
   You must only enter the details of mobile clouds on which you want to run the AFT Suite.
   - BitBar Cloud
     a. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.
     b. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  . **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.

Option	Action
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

c. Click **Apply**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

◦ pCloudy Cloud

a. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

b. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

c. Click **Apply**.

Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

◦ Perfecto Cloud

a. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

b. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

c. Click **Apply**.

Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

4. Click **Apply and Close**.
5. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
6. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
7. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

8. Click **OK**.

Results

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are connected to the multiple mobile clouds from the location you configured.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Configuration of Windows test runs

You can run Windows tests that you created for Windows desktop applications in Rational® Functional TesterHCL OneTest™ UI.

Running a Windows test

To verify that a Windows application works as designed, you must run the recorded Windows test. You can choose to run the Windows test either on the local computer or on a remote computer.

Before you begin

You must have completed the following tasks:

- Recorded a Windows test.
- Ensured that the Scale and layout in the display settings of the system match with the settings used to create the test.

  Note: The display scaling must always be 100%.

- Specified your preference for capturing the screen during the playback from **Windows > Preferences > UI Test Playback > Report** tab, if necessary.

About this task

When you want to run the Windows test on the local computer, you can directly run the Windows test. However, when you want to run on a remote computer, you must first specify the details of a remote computer before you play back the Windows test.

Important: You must not lock the computer screen or minimize the application while you run the Windows test. Otherwise, the playback is interrupted.

Restriction: Although the interactions with some controls within a Microsoft Add-In window (which are not hyperlinks, such as buttons, or inputs) on an embedded browser page are recorded, these actions do not show in the playback of the recorded script.

To workaround this missing action, you must update the test case to ensure that there is a test step to click on the embedded browser prior to the test steps interacting with the controls contained within the Add-in application window.

Some of these Add-In embedded browsers can have multiple containers and you can determine the correct container by using a trial and error method.

You can try the different available containers by highlighting this test step and clicking on the different containers available in the **SmartShot View**, and then right-click and choose **Use this element as the step target**. You can playback the test to check if the interactions are highlighted correctly. If not, you can repeat with a different container until the playback displays the interactions.

Note: Rational® Functional TesterHCL OneTest™ UI can handle a long scroll action effectively during the playback if the property *IsScrollItemPatternAvailable* of the element is set to *True* in the application under test.

1. Perform the following steps based on your choice to run the test:
   - For the local computer, double-click the test, which you want to run, in the **Test Navigator** view.
   - For a remote computer, perform the following steps:
     a. Go to **Windows > Preferences > Test Execution > UI Test Playback > Mobile Device** tab.
     b. Select the **Host** check box.
     c. Specify the following details:
        - The IP address of the remote computer in the **Host** field.
        - Port number of the UI Test Agent that is running on the remote computer in the **Port** field.
     d. Double-click the test, which you want to run, in the **Test Navigator** view.
2. Click **Run Test** in the test editor.

   The test recorded on the Windows application is played back.

Results

You have run the Windows test successfully. After the playback is completed, a unified report is generated for the Windows test.

Related information

Unified reports on page 1455

Configuration of AFT Suite runs

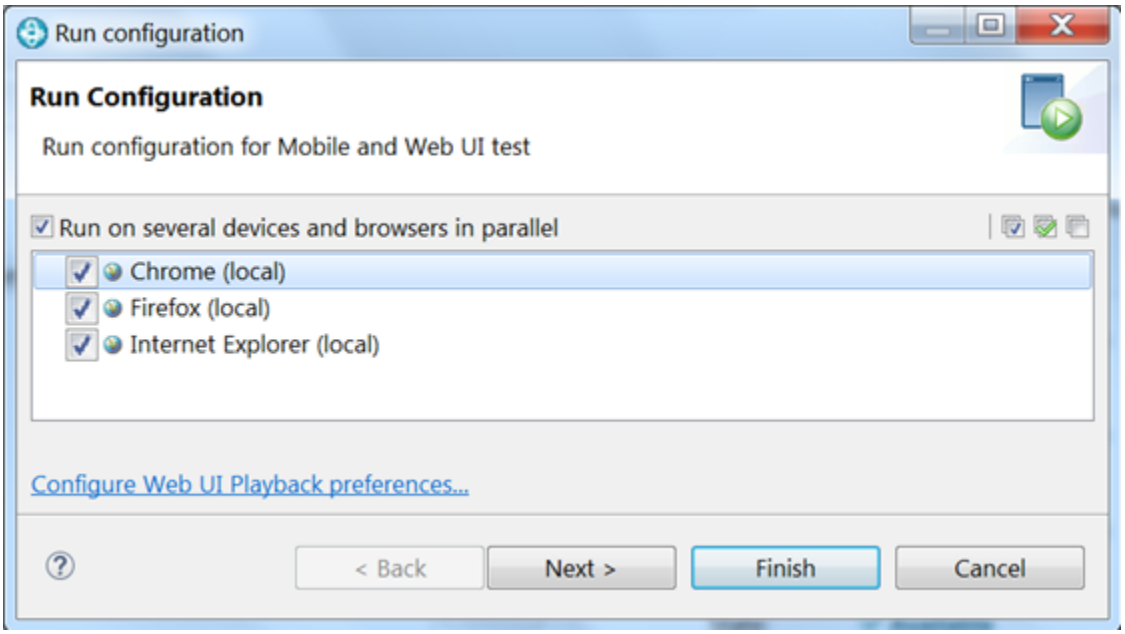
When you create Web UI, mobile, or Windows tests, you must configure the play back of the recorded tests as AFT Suite runs before you can view their test results.

Accelerating the test effort with distributed testing

The Rational® Functional TesterHCL OneTest™ UI UI Test perspective helps you accelerate the test effort by providing ways to distribute test execution across multiple browsers and multiple computers simultaneously.

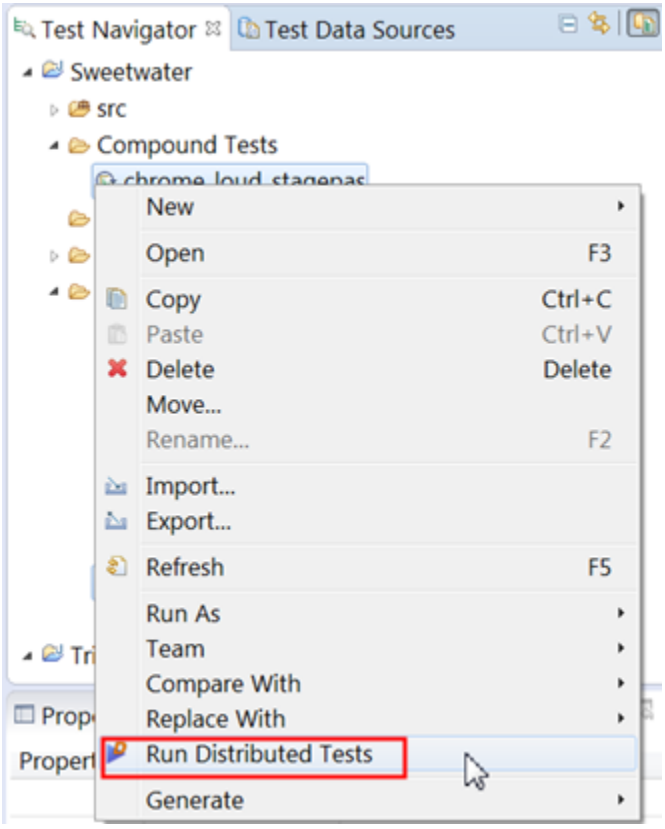
Running a Web UI test on multiple browsers simultaneously

When you run a Web UI test, you can run it on multiple browsers simultaneously and thus achieve browser coverage for that test. You do this by selecting the **Run on several devices and browsers in parallel** check box in the Run Test wizard.



Running multiple Web UI tests on multiple browsers simultaneously

You can further accelerate the test effort by running multiple Web UI tests on multiple browsers and mobile devices simultaneously. You do this by selecting a group of tests in the Test Navigator, including Web UI tests in a compound test, and selecting **Run Distributed Tests** .



For instructions, see Running multiple Web UI and compound tests simultaneously on page 1357.

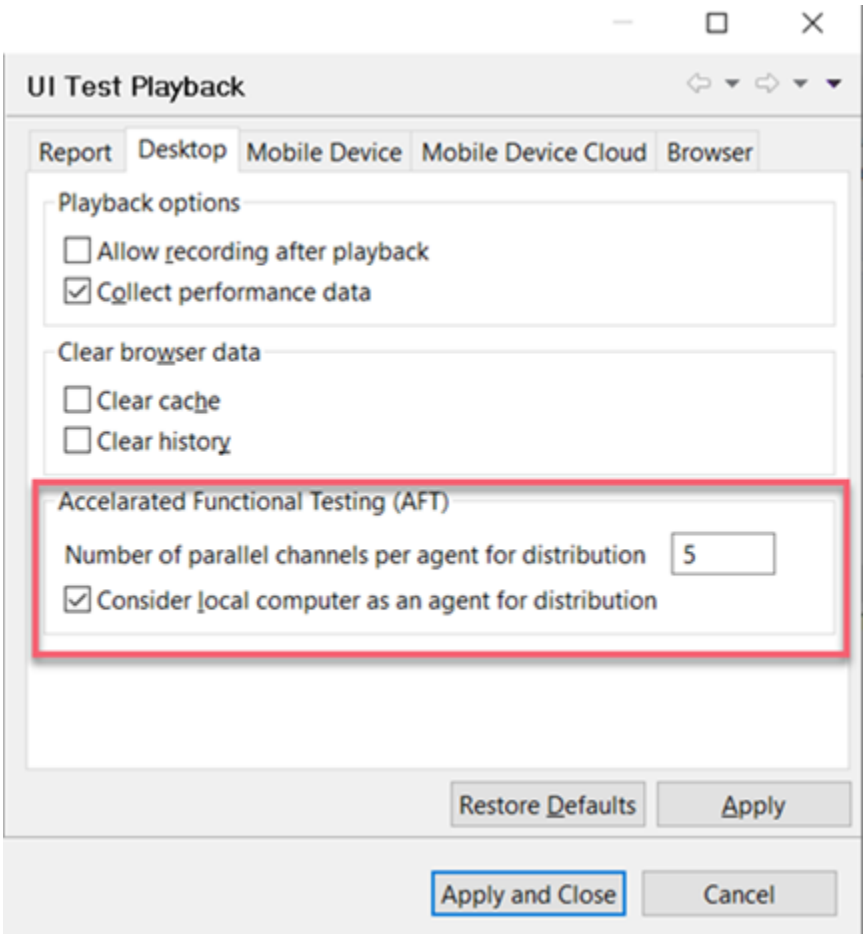
Running multiple Web UI tests on multiple browsers and platforms simultaneously

The next scenario in accelerating the test effort is the same as the previous one, but in this case you also distribute the tests across multiple remote computers. To do this, you install a Rational® Performance TesterHCL OneTest™ Performance agent on each remote computer and point the agent at the Rational® Functional TesterHCL OneTest™ UI. Be sure to specify the host name and port or IP address of the Rational® Functional TesterHCL OneTest™ UI and select the Web UI check box. Also, be sure to shell-share Rational® Performance TesterHCL OneTest™ Performance and Rational® Functional TesterHCL OneTest™ UI. For further instructions, see Installing Rational® Performance TesterHCL OneTest™ Performance agents, especially Step 12.

During test execution, the agent computer is given priority over local execution; that is, tests always run on the agent computer by default if agents are available to the workbench. If no agents are available, tests run on the local computer.

Each remote agent can include several channels (or streams) for running tests, and if the number of selected tests is greater than the number of channels, each channel can contain multiple Web UI tests. The Web UI tests within a channel are run sequentially.

In the Web UI Playback (Desktop) Preferences (**Window > Preferences > Test > Test Execution > UI Test Playback (Desktop) > Desktop**), you can set the number of parallel channels for each agent computer.



You can set the number of parallel channels to the number of virtual users (VUs) that you are licensed for or to any number that is to be associated with this feature. The default value is 5. Limit the number to no more than 15 on a computer with 4 GB of RAM. The selected Web UI tests are pre-arranged into the specified number of channels based on earlier execution times of each test, thus balancing the total execution time of each channel.

Note: If you run compound tests, the number of compound tests multiplied by the number of selected browsers must not exceed the number of channels.

For further information about Rational® Performance TesterHCL OneTest™ Performance agents, see Working with agents. You can download the Rational® Performance TesterHCL OneTest™ Performance agent from Passport Advantage.

For more information about running multiple Web UI tests and compound tests by using a Rational® Performance TesterHCL OneTest™ Performance schedule, see Running tests from a schedule on page 1363.

You can also automate your testing by running the test from the command line interface, by running the test from Rational® Quality Manager, or by running the test with UrbanCode™ Deploy. For more information, refer to the following topics:

- Running a single Web UI test on multiple browsers and devices simultaneously on page 1376
- Running a Web UI test or compound test from the command line on multiple browsers on page 1411
- Running a Web UI test using IBM Rational Quality Manager on page 1361
- Testing with UrbanCode Deploy on page 448

Playing back an Accelerated Functional Test asset

After you create an Accelerated Functional Test (AFT) asset for Web UI or Compound tests, you can play back these tests anytime later by using the AFT XML file.

Before you begin

You must have created AFT test asset for the Web UI or compound tests. See Creating an Accelerated Functional Test asset on page 746

About this task

You can also play back the AFT asset from the command-line.

1. Right-click the accelerated functional asset XML file in the **Test Navigator** pane,and click **Run Distributed Tests**.

   The **Run Accelerated Functional Test** dialog box is displayed.

   You can choose to select the following checkboxes:
   - **Re-run failed tests only the from last playback**: Select this checkbox if you want to rerun only the failed tests from the previous playback.

     Note: If this option is enabled, the failed tests are rerun only on the browsers and location on which the test failed previously.

   - **Fix the browser-driver incompatibility**: Select this checkbox to automatically resolve the incompatibility between the browser and the driver, while you play back the AFT test asset.

     Tip: As the playback starts only after the appropriate driver is downloaded, a timeout error might occur if the application is not started within the time limit specified in the **Time Out** field. You must increase the time in the **Time Out** field. To resolve this error, you can modify the timeout value. The default timeout is 10 seconds. To modify the timeout, check the option and enter a new value.

2. Click **OK**.

Results

The tests mentioned in the AFT XML are played back on the specified browsers, devices, and agents.

Related information

Configuration of test runs from the command line on page 1381

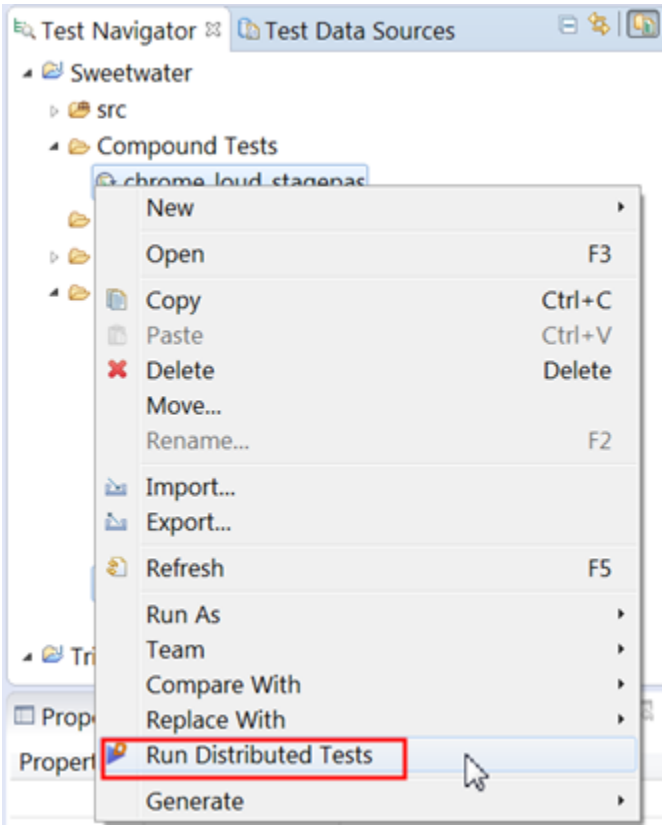
Running multiple Web UI and compound tests simultaneously

To maximize test coverage in the shortest possible time, you can set up distributed testing and run several Web UI tests and compound tests simultaneously on several remote computers, for several operating systems and browsers.

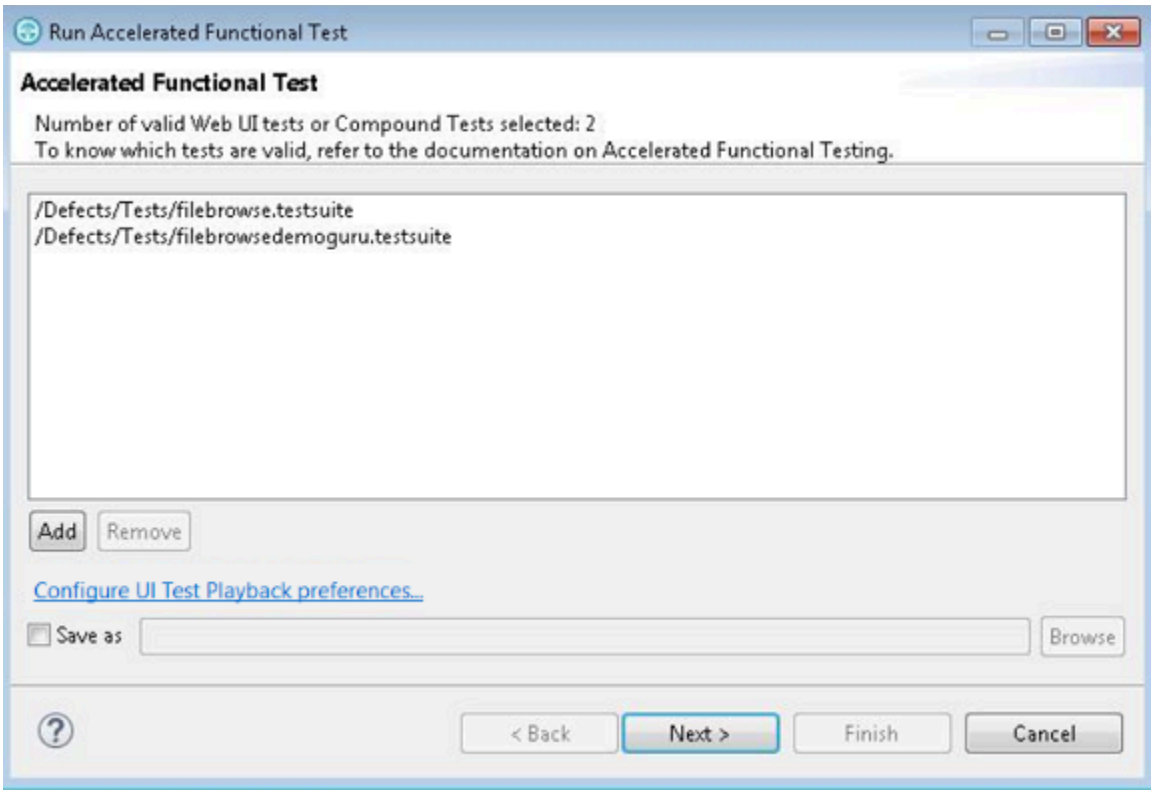
Before you begin

See Accelerating the test effort with distributed testing on page 1352 for requirements for testing on remote agents.

1. In the Test Navigator, right-click a folder that contains multiple Web UI tests, compound tests, or both, and click **Run Distributed Tests**. (Only Web UI tests can be included in the compound tests.) If the tests reside in different folders, use multi-select to select the individual tests before right-clicking. You can also right-click a Project to select all of the Web UI tests in that project.

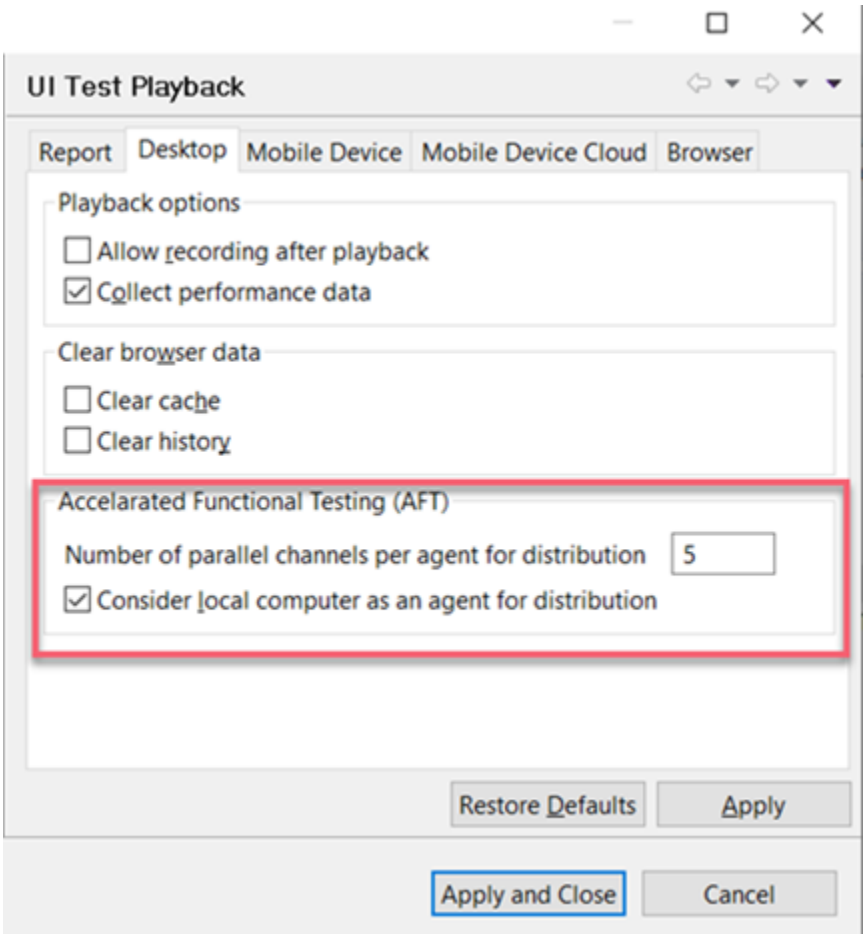


2. In the **Run Accelerated Functional Test** window, review the lists of tests that are queued up for the test run. In the following example, one single test and one compound test are selected. Add or remove any other tests to run and click **Next**.

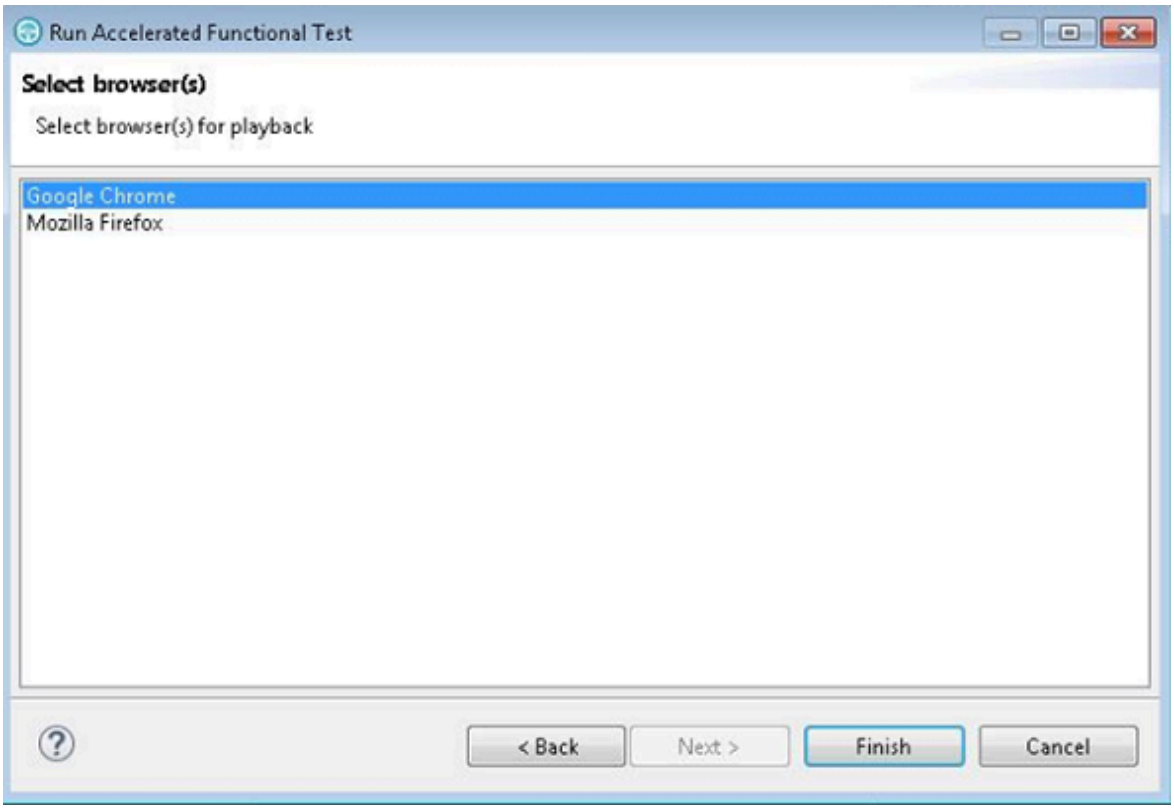


Note: To rerun the same set of tests, select **Save as** check box to save the Web UI or compound tests as a test asset.

If necessary, click **Configure UI Test Playback preferences** to modify the Accelerated Functional Testing (AFT) preferences.



3. Select the browser that will be used for test playback, either Google Chrome or Mozilla Firefox, and click **Finish**.



Result

The selected web browsers open and the test is played back. Do not do any action on the web browsers while the test is playing back. The statistical and live reports show the live data as the test is played back. You do not need to choose the same browser that was used to record the test.

4. Edit the XML file to add more devices, agents, or devicemode. Refer to the following sample file.

```
<?xml version="1.0" encoding="UTF-8"?>
  <inits>
    <group>
      <tests>
        <test path="/WebUProj/ariesweb1.testsuite"/>
        <test path="/WebUProj/ariesweb2.testsuite"/>
        <test path="/WebUProj/ariesweb3.testsuite"/>
        <test path="/WebUProj/ariesweb4.testsuite"/>
        <test path="/WebUProj/ariesweb5.testsuite"/>
      </tests>
      <devices>
          <device id="Emulator:Pixel_API_29"></device>
          <device id="Simulator:iPhone 11 Pro_14.4"></device>
      </devices>
      <browsers>
        <browser name="chrome" id="Apple iPhone 6 Plus"/>
        <browser name="chrome" id="Google Nexus 5"/>
        <browser name="firefox"/>
      </browsers>
      <locations>
```

```
      <location      host="9.113.29.29"/>
      <location      host="9.113.29.30"/>
      <location      host="9.113.29.31"/>
      <location      host="9.113.29.32"/>
      <location      host="civcez228.company1.com"/>
    </locations>
  </group>
  <group>
    <tests>
      <test path="/WebUProj/ariesweb6.testsuite"/>
    </tests>
    <browsers>
<browser name="chrome" devicemode="Apple iPhone6 Plus" headless="true"/>
<browser name="firefox"/>
    </browsers>
    <locations>
<location host="localhost"/>
    </locations>
 </group>
</inits>
```

5. If you are running tests on emulators and device clouds, edit the XML file to add the following values:

```
    <browsers>
      <browser name="<name_of_the_browser>" id="Emulator: <device_name>"/>
      <browser name="<name_of_the_browser>" id="<device_cloud_name>: <device_name>"/>
     </browsers>
```

For example,

```
    <browsers>
      <browser name="Chrome" id="Emulator:Google_Pixel_7_Pro"/>
      <browser name="Chrome" id="BrowserStack : Google Pixel 7 Pro"/>
    </browsers>
```

Results

After the test run completes, there is a single UI Test report for all the tests. To view a functional report, you must generate it manually by right-clicking a report in the Results folder and clicking **Generate Functional Test Report**. The Resources tab in the statistical report is empty because a Web UI test does not monitor resources.

Related information

Running a Web UI test or compound test from the command line on multiple browsers on page 1411

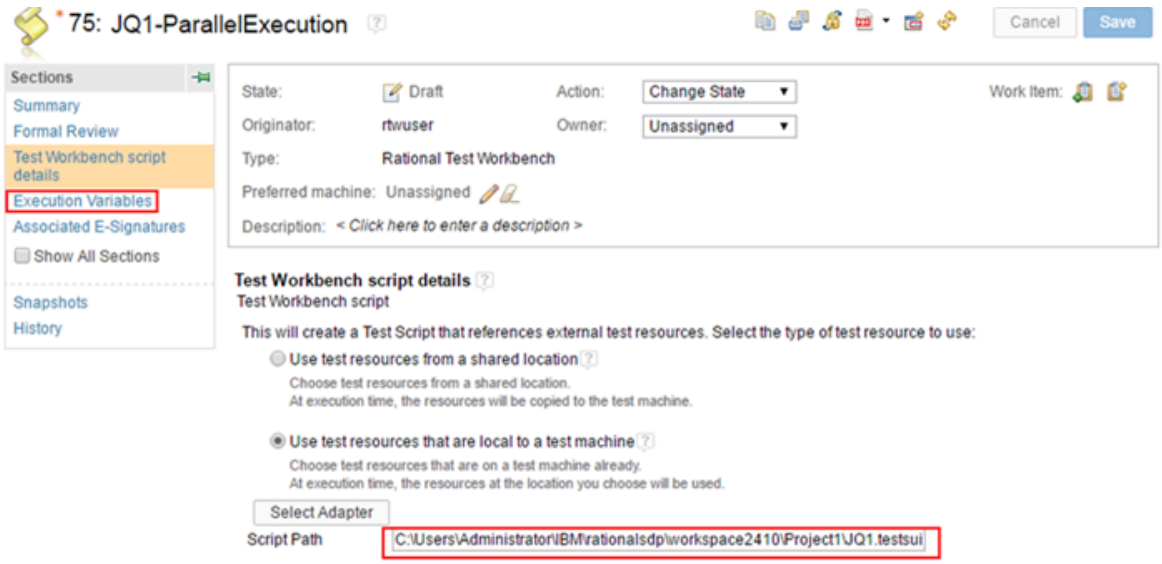
Running a Web UI test using IBM® Rational® Quality Manager

Another way to automate your testing is to use IBM® Rational® Quality Manager. With Rational® Quality Manager, you can run an individual Web UI test or specify a particular browser to run the test on. You can also accelerate your testing by running the test on all browsers and mobile devices simultaneously or on a selected set of browsers.

About this task

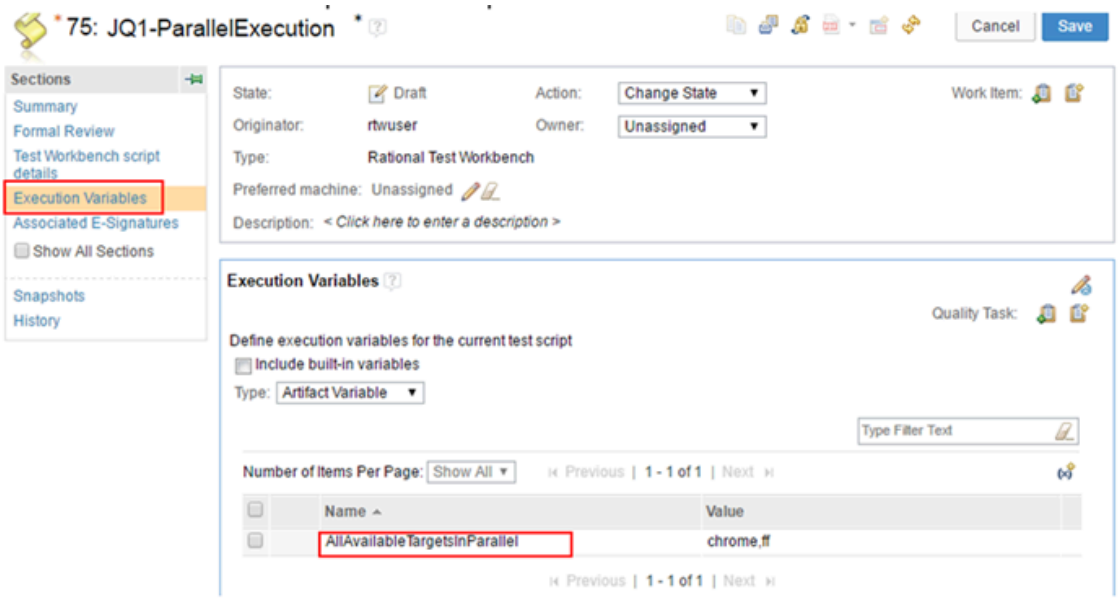
To run a test from Rational® Quality Manager you first define a test in RQM that includes the path to the Web UI test. You then configure and run the Rational® Quality Manager adapter that is installed by default when you install Rational® Functional TesterHCL OneTest™ UI.

1. Using the **Test Workbench script details** page of a Rational® Quality Manager test script, associate a Web UI test script with the Rational® Quality Manager script as shown below:



   Be sure to specify the value in the Script Path field to be the path to the Web UI test script. For additional details, see Creating a reference to an automated test script on a local test machine.

2. To run the test on a set of browsers or on all browsers and connected mobile devices, go to the Execution Variables page of the Rational® Quality Manager test script and specify the execution variable to pass to the script.



a. To run the test on all available browsers and connected mobile devices, use the execution variable `AllAvailableTargetsInParallel` and specify the value as `All`.

b. To run the test on a selected set of browsers, use the execution variable `AllAvailableTargetsInParallel` and specify the value as a comma-separated list of browsers, for example, `chrome, ff`.

c. To run the test on a particular browser, use the execution variable `RTW_WebUI_Browser_Selection` and specify the browser as the value, for example, `chrome`.

Note: You can also use `RTW_WebUI_Browser_Selection` as a test variable in the Rational® Functional TesterHCL OneTest™ UI UI Test perspective. For details, see Defining a variable to run a test with a selected browser on page 578.

3. Save the test and run it using the Rational® Quality Manager adapter.

Related information

Running a Web UI test or compound test from the command line on multiple browsers on page 1411

Running tests from a schedule on page 1363

Running tests from a schedule

To run multiple Web UI tests in parallel on different browsers, you can add the test to an IBM® Rational® Performance TesterHCL OneTest™ Performance schedule.

Before you begin

- You must install IBM® Rational® Performance TesterHCL OneTest™ Performance and IBM® Rational® Functional TesterHCL OneTest™ UI into the same instance of Eclipse and use a single workspace (also called shell sharing).
- To run a Web UI test on a remote computer, you must install Rational® Performance TesterHCL OneTest™ Performance Agent on that computer. While installing the agent, ensure that you select the **The agent will be used primarily to support remote execution of Web UI tests** check box.

1. In the **Web UI Test** perspective, create a Web UI test and ensure that you run it successfully at least once before adding to a schedule.
2. Open the **Performance Test** perspective and create a schedule.
3. To add a Web UI test, in the Schedule editor, click **User Group 1** and click **Add > Test**.
4. Select a test and click **OK**.
5. To specify a variable:

    a. Click **User Group 1** and from the User Group Details area click **Variable Initialization**.

    b. Click **Add**.

    c. For **Variable Name**, type `RTW_WebUI_Browser_Selection`, and for **Initial value** type `Firefox`, `Chrome`, `Internet Explorer`, or `Internet Explorer 64`. This action specifies which web browser to use for the user group.
    You can add multiple user groups and assign a different web browser to each user group.

    Note: You can also add the variable in the Web UI test itself. See Defining a variable at the test level on page 578. If the variable name is the same in the test and schedule and the **Visible in** field in the test is set to **All tests for this user**, the variable defined at the schedule level is used when the schedule is run.

6. **Optional:** Run a user group at a remote location where Rational® Performance TesterHCL OneTest™ Performance Agent is installed.
7. **Optional:** To generate the UI Test report, select a User Group, click the **Options** tab and click the **Edit options** button. In the **UI Test** tab, click the **Enable UI Test reports** check box. To capture screenshots of the user interface that was recorded, click **Enable screenshots capture**
    When you run a Web UI test as part of a schedule with virtual users, by default, the UI Test report is not generated because the report will contain lot of data for each virtual user. There are logs and statistical reports for analysis. However, the UI Test report might be useful to you if you are running a schedule on a control agent with a single user.

    Note: . The **UI Test** tab shows up only if there is a mobile or a Web UI test in the schedule.

8. Save the schedule. To run it, click **Run Schedule**.

Running Web UI tests as an AFT Suite on BrowserStack Cloud

After you create an AFT Suite for Web UI tests, you can run the AFT Suite on devices connected to the BrowserStack Cloud to accelerate the test efforts.

Before you begin

You must have completed the following task:

- Created an AFT Suite with Web UI tests to run on the BrowserStack Cloud. See Creating an AFT Suite to run Web UI tests on BrowserStack Cloud on page 747.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Select the project that contains the AFT Suite in the **Test Navigator** pane.
   The AFT Suite that you created for the Web UI tests is located in the **Accelerated Functional Tests** folder.
3. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
4. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

   Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

5. Click **OK**.

Results

You have run the Web UI tests that are in an AFT Suite. The tests run on mobile devices that are connected to the cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report and choose to export the report. See Unified reports on page 1455.

Running mobile tests as an AFT suite

After you create an Accelerated Functional Test (AFT) suite for mobile tests, you run the AFT suite by using the **Run Distributed Tests** option.

Before you begin

You must have completed the following tasks:

- Created an AFT suite with mobile tests. See Creating an AFT suite for mobile tests on page 749.
- The configured UI Test Agent must be running.

1. Right-click the XML file in the **Test Navigator** pane, and then click **Run Distributed Tests**.

   The **Run Accelerated Functional Test** dialog box is displayed.

   The following options are displayed in the **Run Accelerated Functional Test** dialog box:
   - **Re-run failed tests only from the last playback**: Select this checkbox if you want to rerun the failed tests only from the last playback. The failed test is again run on the same device and location on which it previously failed.

     Note: This option is enabled only if the test was already run at least once.

   - **Fix the browser-driver incompatibility**: This option is not applicable for mobile tests and it is disabled.
2. Click **OK**.

Results

You have run mobile tests as an AFT suite.

Related information

Running mobile tests for Android applications from the command-line on page 1415

Running mobile tests for iOS applications from the command-line on page 1424

Running mobile tests as an AFT Suite on BitBar Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to BitBar Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the BitBar cloud on page 754.
- Set up your account to access the BitBar Cloud. You must have valid credentials such as the host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
- Uploaded the `.ipa` or `.app` file of the iOS application or the `.apk` file of the Android application that you want to test in the BitBar Cloud. For information, refer to Live Testing in the BitBar documentation.
- Installed the Android and iOS applications that you want to test on the device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

a. Click the **Mobile Device Cloud** tab.

b. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.

c. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  .  **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

d. Click **Apply and Close**.

 **Note:** When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on BrowserStack Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on multiple devices connected to the BrowserStack Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on BrowserStack Cloud on page 757.
- Set up your account to access the BrowserStack Cloud. You must have valid credentials such as the user name, either the host name or the URL of the BrowserStack cloud instance, and an API key to authenticate the connection. This key is available under **ACCESS KEY** in the BrowserStack dashboard.
- Uploaded the `.ipa` or `.app` file of the iOS application or the `.apk` file of the Android application to the BrowserStack server for testing. For information, refer to Live Testing in the BrowserStack documentation.
- Installed the Android and iOS applications that you want to test on the device.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **BrowserStack host** checkbox to enable the options on the **BrowserStack Device Cloud Environment** panel.

c. Perform the actions as listed in the following table:

Option	Action
Required	
BrowserStack host	Enter the URL of the BrowserStack cloud instance.
User Name	Enter the user name of your BrowserStack account.
API key	Enter the API key of your BrowserStack Cloud account to authenticate the connection.
Optional	
Project	Enter an appropriate name for the project with which you can identify the test run on the BrowserStack Cloud dashboard.

d. Click **Apply and Close**.

Note: When the connection with the BrowserStack Cloud instance is successful, the mobile devices that are on the BrowserStack Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

What to do next

You can view the test result that is displayed as a unified report for mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on pCloudy Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to pCloudy Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the pCloudy cloud on page 760.
- Set up your account to access the pCloudy cloud. You must have valid credentials such as the host name or the URL of the pCloudy cloud instance, the user name, and an API key to authenticate the connection.
- Uploaded the `.apk` file of the Android application or the `.ipa` file of the iOS application that you want to test in the pCloudy Cloud. For more information, refer to Upload Android App or Upload iOS App in the pCloudy documentation.
- Installed the Android application or iOS application that you want to test on the device. For more information, refer to Installing Android app or Installing iOS app in the pCloudy documentation.
- Verified that the device you want to test is available and in the *Released* state on the pCloudy cloud.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

d. Click **Apply and Close**.

Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on Perfecto Cloud

After you create an AFT Suite for mobile tests, you can run the AFT Suite on devices connected to Perfecto Cloud to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT Suite with mobile tests. See Creating an AFT Suite to run mobile tests on the Perfecto cloud on page 763.
- Set up your account to access the Perfecto Mobile Cloud. You must have valid credentials such as the host name or the URL of the Perfecto Mobile Cloud instance and the security token to authenticate the connection.

- Uploaded the `.ipa` file of the iOS application or the `.apk` file of the Android application that you want to test in the Perfecto Cloud. For information, refer to Manage Apps in the Perfecto documentation.
- Installed the Android and iOS applications that you want to test on the device.

About this task

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Perform the following steps in the **UI Test Playback** pane:

   a. Click the **Mobile Device Cloud** tab.

   b. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

   c. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

   d. Click **Apply and Close**.

   Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

4. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
5. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
6. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

7. Click **OK**.

Results

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are on the Cloud from the location you configured.

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

What to do next

You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running mobile tests as an AFT Suite on multiple mobile clouds

After you create an AFT Suite for mobile tests, you can run the tests simultaneously on devices connected to multiple mobile clouds to accelerate the test efforts.

Before you begin

You must have completed the following tasks:

- Created an AFT suite with mobile tests, See Creating an AFT Suite to run the tests on multiple mobile clouds on page 767.
- You must have set up accounts and obtained valid credentials for more than one mobile cloud as mentioned below:
  - BitBar Cloud: The host name or the URL of the BitBar Cloud instance, and an API key to authenticate the connection.
  - pCloudy Cloud: The host name or the URL of the pCloudy Cloud instance, the user name, and an API key to authenticate the connection.
  - Perfecto Cloud: The host name or the URL of the Perfecto Cloud instance, and the security token to authenticate the connection.
- Uploaded the `.ipa` file of the iOS application or the `.apk` file of the Android application that you want to test in the mobile cloud. For more information, refer to the following:
  - BitBar: Live Testing in the BitBar documentation.
  - pCloudy: Upload Android App or Upload iOS App in the pCloudy documentation.
  - Perfecto: Manage Apps in the Perfecto documentation.
- Installed the Android or iOS application that you want to test on the device.

Note: To install the application in a device on the pCloudy Cloud, refer to Installing Android app or Installing iOS app in the pCloudy documentation.

- If the test to be run on pCloudy Cloud, verified that the device you want to test is available and in the Released state on the pCloudy cloud.

About this task

To run the AFT Suite on multiple clouds, you must configure the mobile clouds with IBM® Rational® Functional TesterHCL OneTest™ UI. You need to configure only those mobile clouds, to which the devices you want to run the tests are connected.

1. Open the **UI Test** perspective in Rational® Functional TesterHCL OneTest™ UI, if it is not already open.
2. Click **Window > Preferences > Test > Test Execution > UI Test Playback**.

   The **UI Test Playback** pane is displayed.
3. Click the **Mobile Device Cloud** tab, and perform the following steps to configure the mobile clouds with Rational® Functional TesterHCL OneTest™ UI:
   You must only enter the details of mobile clouds on which you want to run the AFT Suite.
   - BitBar Cloud
     a. Select the **BitBar host** checkbox to enable the options on the **BitBar Device Cloud Environment** panel.
     b. Perform the actions as listed in the following table:

Option	Action
BitBar host	Enter the host name of the BitBar Cloud instance.
API Key	Enter the API key of your BitBar Cloud account to authenticate the connection, and then click the **Refresh projects and device groups**  . **Note:** Clicking the **Refresh projects and device groups**  enables the **Project** and **Device Group** fields.
Project	Select the BitBar project from the drop-down list.
Device Group	Select the mobile device group that you want to use in the BitBar cloud.

Option	Action
Test Run	Enter an appropriate name for the test with which you can identify the test run on the BitBar Cloud dashboard.

c. Click **Apply**.

Note: When the connection with the BitBar Cloud instance is successful, the mobile devices that you have configured on the BitBar Cloud are displayed in the **Run using** drop-down list.

◦ pCloudy Cloud

a. Select the **pCloudy host** checkbox to enable the options on the **pCloudy Device Cloud Environment** panel.

b. Perform the actions as listed in the following table:

Option	Action
pCloudy host	Enter the host name of the pCloudy instance.
User Name	Enter the user name of your pCloudy account.
API Key	Enter the API key of your pCloudy account to authenticate the user name for a successful connection.

c. Click **Apply**.

Note: When the connection with the pCloudy instance is successful, the mobile devices that you have configured on the pCloudy Cloud are displayed in the **Run using** drop-down list.

◦ Perfecto Cloud

a. Select the **Perfecto host** checkbox to enable the options on the **Perfecto Device Cloud Environment** panel.

b. Perform the actions as listed in the following table:

Option	Action
Perfecto host	Enter the host name of the Perfecto Cloud instance.
Security Token	Enter the Token to authenticate the connection to the Perfecto Cloud instance.

c. Click **Apply**.

Note: When the connection with the Perfecto Cloud instance is successful, the mobile devices that you have configured on the Perfecto Cloud are displayed in the **Run using** drop-down list.

4. Click **Apply and Close**.
5. Select the project that contains the AFT Suite, in the **Test Navigator** pane.
   The AFT Suite that you created for the mobile tests is located in the **Accelerated Functional Tests** folder.
6. Expand the contents of the **Accelerated Functional Tests** folder to view the AFT XML that is displayed with the name you provided.
7. Right-click the XML file of the AFT Suite, and then click **Run Distributed Tests**.
   The **Run Accelerated Functional Test** dialog is displayed.

Note: If you repeat the AFT Suite run because the previous run failed, the **Re-run failed tests only from the last playback** option is available for selection. When you select this option, the AFT Suite is run on the devices and location on which it previously failed.

8. Click **OK**.

Results

The test result is displayed as a unified report in Rational® Functional TesterHCL OneTest™ UI.

You have run the mobile tests that are in an AFT Suite. The tests run on mobile devices that are connected to the multiple mobile clouds from the location you configured.

What to do next

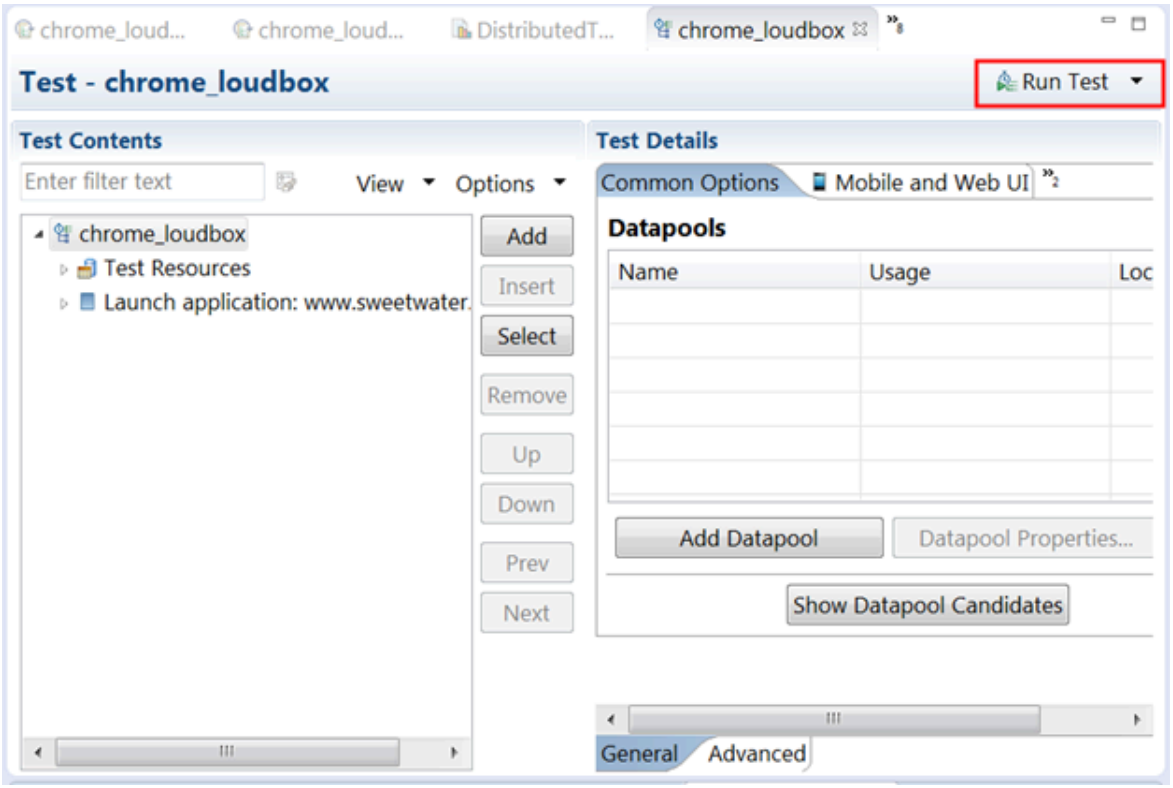
You can view the unified report for the mobile tests and choose to export the unified report. See Unified reports on page 1455.

Running a single Web UI test on multiple browsers and devices simultaneously

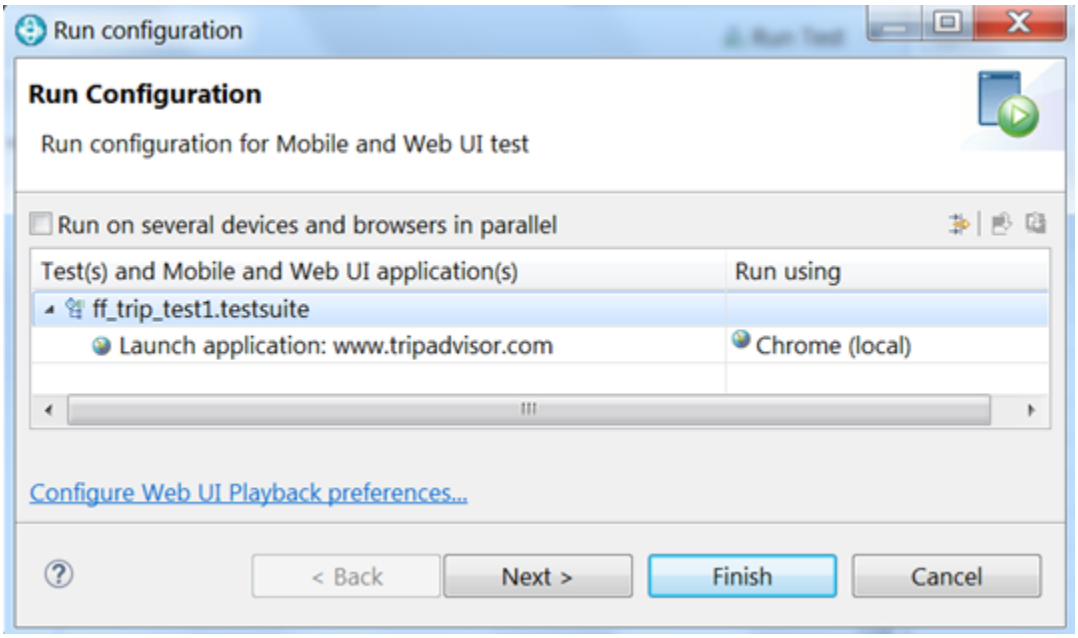
Rather than run a Web UI test on one browser at a time, you can run a single Web UI test on multiple browsers simultaneously. Doing so can significantly speed up your test effort. You can also extend your test coverage by

adding mobile devices to the test run. You can run the test from Rational® Functional TesterHCL OneTest™ UI, from the command line, or from Rational® Quality Manager. This topic describes how to run a test from the UI Test perspective.

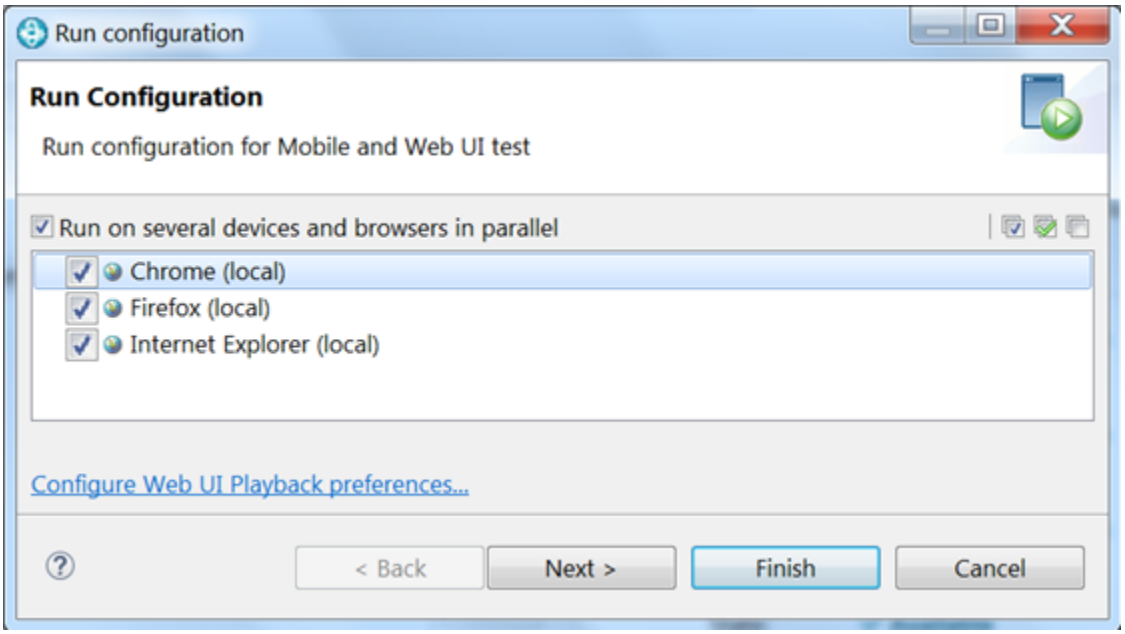
1. To open a test, double-click it in the Test Navigator view. The test opens in the Test Editor.



2. In the Test Editor, click **Run Test** to open the Run Configuration window.



3. Select **Run on several devices and browsers in parallel**.

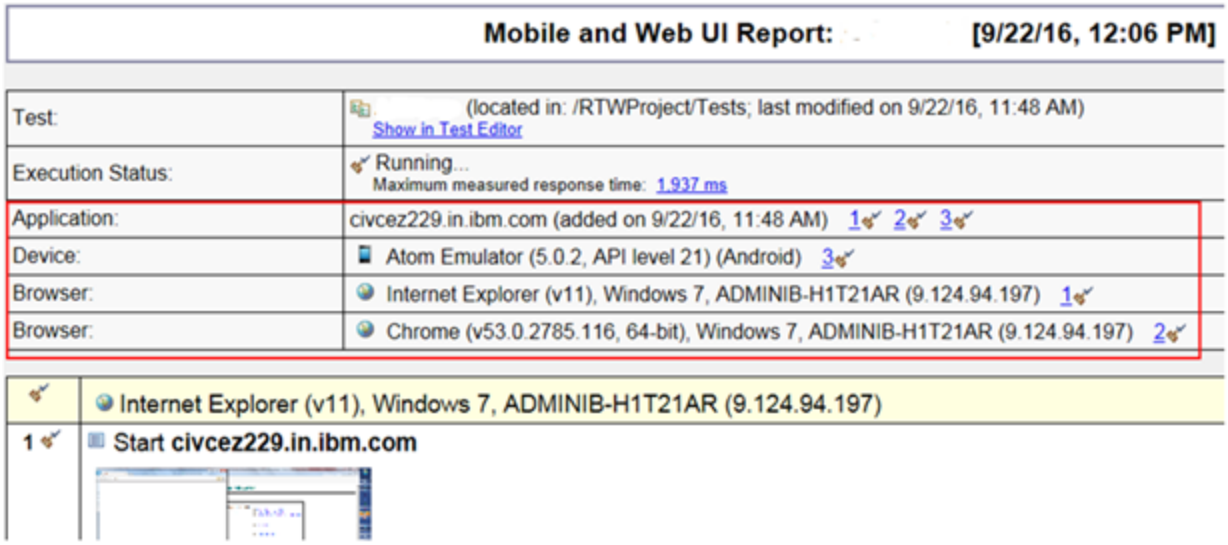


   You can now choose to run the test in parallel using any or all browsers, devices, and emulators that are listed. All supported web browsers that are installed on your computer are displayed. To run a test on a mobile device or emulator, the device must be connected and must be in passive mode.

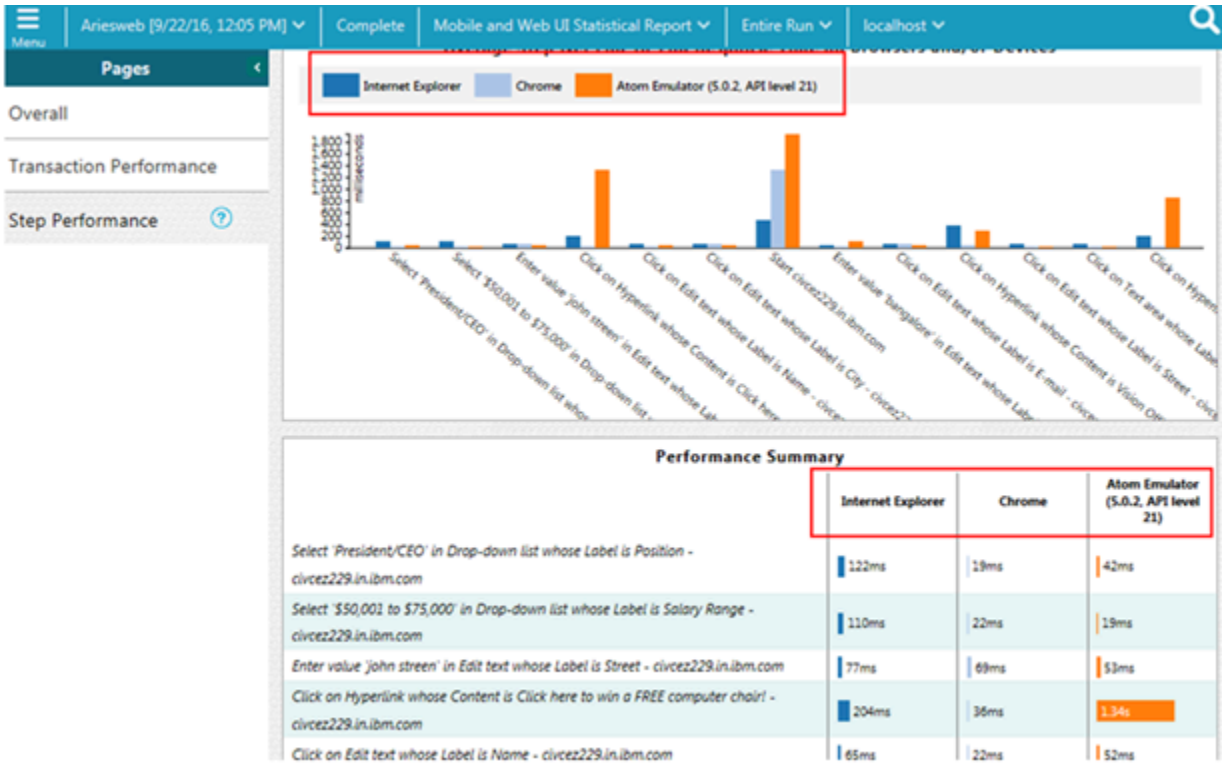
4. Click **Finish**.

Results

By default, a **UI Test Report** is displayed automatically in real-time during a test run. The report displays the name and location of the test, the test execution status, the Web application under test, the duration of the test, and finally, each step in the test. When you run the test in parallel across multiple browsers and mobile devices, you see a single, consolidated report that lists each browser and device on which the test was run.



In addition, you can learn how long it took each step to run on each browser and device by viewing the **UI Test Statistical Report** and clicking the Step Performance tab. For example, you can see in the Performance Summary shown below that the first step took 122ms to run in Internet Explorer, 19 ms to run in Chrome, and 42 ms to run on the Android Atom Emulator. You can use this information to help you compare the performance of the application under test across different browsers.



The UI Test Statistical Report is displayed at the end of the test run in a separate tab from the UI Test Report. The UI Test Statistical Report is displayed in the Test Execution perspective.

Related information

Running a Web UI test on page 1243

Running a Web UI test or compound test from the command line on multiple browsers on page 1411

Evaluating desktop Web UI results on page 1471

Viewing On App and Off App response time on page 1478

Setting a global delay for test steps

You can set the duration of a global delay in milliseconds between all the steps instead of adding the wait time manually for each of the steps in a recorded test. You can specify this global delay as a preference for the playback of Web UI, Windows, or mobile tests.

Before you begin

You must have recorded a Web UI, Windows, or mobile test. See Testing in the UI Test perspective on page 532.

About this task

When you set the delay, during play back, each step starts after the previous step is completed and the delay time is passed. The global delay setting applies to all the test runs including the test runs by using the command-line interface.

You can also set the delay from the **Run Configuration** dialog by clicking the **Configure UI Test Playback Preference** checkbox, and then by entering the value in milliseconds. The value specified in this dialog applies only to the current test and it overrides the value that is set on the **Preferences** page.

1. Go to **Window > Preferences > Test Execution > UI Test Playback**.
2. Click the **Desktop** tab.
3. Select the **Global steps delay in milliseconds** checkbox in the **Playback options** section.

   Note: You must clear the **Global steps delay in milliseconds** checkbox to disable the global delay for all the test runs.

4. Click **Apply and Close**.

Results

You have set a global delay for the test steps.

Related information

Running a Web UI test on page 1243

Configuration of test runs from the command line

When you need to run Web UI, mobile, and windows tests using the command line without opening the desktop client to run these tests, you can use the command line of Rational® Functional TesterHCL OneTest™ UI.

Overview

Exemple

Rational® Functional TesterHCL OneTest™ UI supports the usage of command line for certain tasks. To use the command line, you must go to the following directory:

- `<Install_Directory>\cmdline` directory that contains `cmdline.sh` on Linux operating system.
- `<Install_Directory>\cmdline` directory that contains `cmdline.sh` on Mac operating system.
- `<Install_Directory>\cmdline` directory that contains `cmdline.bat` on Windows operating system.

After navigating to the directory, you can run the cmdline file. The command line supports a set of options. You must use the cmdline command with the supported options. You can enter cmdline -help in the command line to view all the supported options.

To run the tests from the command line, you must specify the path of the tests along with the command as follows:

cmdline -workspace <workspace_full_path> -project <project_relative_path> -suite <suite_relative_path>

For example:

cmdline -workspace “D:\My Workspace” -eclipsehome "C:\Program Files\IBM\SDP" -plugins "C:\Program Files\IBM\IBMIMShared\plugins" -project myProject -suite mytestsuitecmdline -workspace “D:\My Workspace” -eclipsehome "C:\Program Files\HCL\HCLOneTest" -plugins "C:\Program Files\HCL\HCLOneTest\plugins" -project myProject -suite mytestsuite

Note:

The -workspace option is followed by a value that contains a space. If the value contains space, then you must enclose the value, D:\My Workspace within quotes. Otherwise, you can provide the value without quotes.

In addition to the supported options, you can create and use varfiles and config files as options. The files can contain specific paths and parameters for running tests on connected systems.

Using a configuration file

You can run tests from the command line by using a configuration file. Each command line option must be followed by an appropriate value.

The contents of a sample configuration file, `config_file1` are as follows:

```
workspace=D:\My Workspace
eclipsehome=C:\Program Files\IBM\SDP
plugins=C:\Program Files\IBM\IBMIMShared\plugins
project=myProject
suite=mytestsuite
```

```
workspace=D:\My Workspace
eclipsehome=C:\Program Files\HCL\HCLOneTest
plugins=C:\Program Files\HCL\HCLIMShared\plugins
project=myProject
suite=mytestsuite
```

To create a config file using the desktop client, see Creating a command-line config file on page 1400.

To run tests from the command line by using the sample config file `config_file1`, you must use the following command:

cmdline –configfile <config file path>

For example:

cmdline –configfile E:\Workspace1\Project1\Tests\config_file1.txt

Using a variable file

Variable file is an xml file with the `.varinit` extension that contains the variable names and values as pairs that are required to connect to the system for running tests.

This is a sample XML file as a variable file that specifies the browser to use on the computer on which you have installed Rational® Functional TesterHCL OneTest™ UI:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<browser>" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

For example, if you connect Emulator:Pixel_2_API_28 as the mobile emulator and you want to use the Chrome browser to run the test, the variable file can be as follows:

```
<?xml version="1.0" encoding="UTF-8"?><inits>
<variable_init value="Chrome(Emulator:Pixel_2_API_28)" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

To run tests from the command line by using a variable file, you must use the following command:

cmdline –varfile <variable file path>

For example:

cmdline –varfile E:\Workspace1\Project1\Tests\var1.varinit

Command line options

Each command line option is used in combination with the other options to accomplish a specific task. You must read and be familiar with the supported options and mandatory options that are required for each task. The following table lists all the supported options with description:

Option	Description
Required	
-project	The path, including the file name of the project relative to the workspace.
-work-space	The complete path to the Eclipse workspace.
-suite	The path, including the file name of the test relative to the project. A test can be a Web UI test, compound test, or an Accelerated Functional Test. In a command, it is mandatory to use one of the following options: • -suite • -aftsuite **Note:** You must provide the file name along with the file extension if you are using an Accelerated Functional Test suite. You must not use the -suite option along with the other options. Starting from 9.2.1.1, you can execute multiple tests simultaneously.

Option	Description
	For example, **-suite** *test1:test2:test3*. You can run any or all tests that are in a folder by using the wildcard character * enclosed within quotes in different combinations in the command line. The following command runs the Web UI tests, compound tests, and Accelerated Functional Tests in the project folder: cmdline -workspace <workspace_full_path> -project <project_relative_path> -suite "*" For example, you can run the following commands: • cmdline -workspace “workspace” -project myProject -suite "*" to run all the tests in **myProject**. • cmdline -workspace “workspace” -project myProject -suite "test*" to run all the tests in **myProject** that have the file names starting with 'test'. • cmdline -workspace “workspace” -project myProject -suite "Compound Tests/*" to run all the tests in the **Compound Tests** folder under **myProject**. • cmdline -workspace “workspace” -project myProject -suite "Web*.testsuite:!Web2" to run all the tests in **myProject** that have the file names starting with 'Web', but not 'Web2' • cmdline -workspace “workspace” -project myProject -suite "Web*.testsuite:/myProject2/Test1*.testsuite" to run all the tests in **myProject** that have the file names starting with 'Web' and all the tests in **myProject2** that have the file names starting with 'Test1'
Optional	
-aftsuite	The path, including the file name of the xml file to run an AFT test. The **aftsuite** option accepts aft XML as the parameter value. It supports only one aft XML as input. For example, **aftsuite**="*aftinput*" You can run the following command to run all the AFT tests that have file names starting with 'AFT': cmdline -workspace <workspace_full_path> -project <project_relative_path> -aftsuite "AFT*" For example, cmdline -workspace "workspace" -project "UI1" -aftsuite "AFT*" You must not use the **-aftsuite** option along with the other options.
-compare	The path, including the file name of the xml file to run an AFT test. The **aftsuite** option accepts aft XML as the parameter value. It supports only one aft XML as input. For example, **aftsuite**="*aftinput*" You must not use the **-aftsuite** option along with the other options.
-configfile	You can use this option to specify the complete path to a file that contains the parameters for a test run. Each parameter must be on a single line. To create a configuration file, you must use an editor that does not wrap lines. Any parameter, whether required or optional, can be set in the configuration file. The command line parameters override the values in this file.

Option	Description
	Notes: • If you are creating a config file manually, the file must be in the UTF-8 format. You must not use quotation marks in this file even for values that contain spaces. • You can create command line config file from the desktop client, which you can use while running tests from the command-line interface or Maven. See Creating a command-line config file on page 1400. This option is available only for Web UI and compound tests.
-eclipse-home	You can use this option to provide the complete path of `eclipse.exe`. For example, `C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest`
-exec-summary	You can use this option to export all of the reports for the test run in a printable format, also known as an executive summary, to the local computer. You must specify the path to store the executive summary.
-exec-summaryreport	You can use this option to export a specific report as an executive summary for the test run to the local computer. You must specify the ID of the report to export as **-execsummaryreport <reportID>**. For example, to export an HTTP performance report, specify `http`. **Note:** You must use this option along with **-execsummary**. To copy the report IDs list into your command line, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. Under **Select reports to export**, select the required reports, and click **Copy ID to clipboard**. You can then paste the clipboard content on to your command line editor.
-exportlog	You can use this option to specify the file path to store the exported test log. Starting from 10.0.1, by using the **-exportlog** parameter, you can provide multiple parameter entries when running multiple tests. You must use a colon to separate the parameter entries. For example: **-exportlog** *c:/logexport.txt:c:/secondlogexport.txt* If there are multiple **-suite** option entries with a single **-exportlog** parameter entry, then the **-exportlog** option generates the appropriate number of test logs by appending 0, 1, 2, and so on to the **-exportlog** option entry name. For example: **-suite** *"sampletest1:sampletest2:sampletest3"* **-exportlog** *c:/logexport.txt* The command generates the following test logs:

Option	Description
	• logexport_0.txt • logexport_1.txt • logexport.txt The last test log generated has the same name as that of the initial **-exportlog** entry. **Note:** If there are multiple **-suite** and **-exportlog** parameter entries, the number of **-suite** entries must match with the number of **-exportlog** entries. Otherwise, the following error message is displayed: `Error, number of -suite and -exportlog entries do not match.`
-export-Report	You can use this option to export the unified report of UI tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. The command syntax is as follows: exportReport "type=<reporttype>;format=<file type1,file type2,file type3>;folder<destination folder path>;filename=<name of the exported file>" For example, to export the report to only the pdf format, you can use exportReport "type=unified;format=pdf;folder=Exportedreport102;filename=testreport **Restriction:** • The icons of the applications under test are not supported in the PDF file. • The text in a table that contains the greater than symbol **>** is printed as `&gt` in the generated PDF of the unified report. • The special characters such as /n, /r, or /t in the step descriptions are replaced with an empty string before the export of the report. • The special characters such as symbols in the step descriptions are replaced with space characters before the export of the report. If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use

Option	Description
	exportReport "type=unified;format=pdf,xml,html;folder=Exportedreport102;filename=testreport You can exclude the **filename** attribute in the **exportReport** option so that the reports are exported with the respective test suite name. For example, when you run multiple tests by using a single command, if you do not specify any file name in **exportReport**, then the name of the test is used for the exported report name, which makes it easy to identify reports.
-exportstatreportlist	You can use this option to specify a comma-separated list of report IDs along with **-exportstats** or **-exportstatshtml** to list the reports that you want to export in place of the default reports, or the reports selected under **Preferences**. To view this setting, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. To copy the report IDs list into your command line, navigate to **Window > Preferences > Test > Performance Test Reports > Export Reports**. Under **Select reports to export**, select the required reports, and click **Copy ID to clipboard**. You can then paste the clipboard content on to your command line editor.
-exportstats	You can use this option to export reports in comma-separated values (CSV) format, with the file name derived from the report name. This directory can be relative to the project or a directory on your file system. If the **-exportstatreportlist** option is not specified, the reports specified on the **Export Reports** page of the **Performance Test Report** preferences are exported.
-exportstatsformat	You can use this option to specify a format for the result that you want to export along with the **-exportstats** option. You must use at least one of the following parameters with the **-exportstatsformat** option: • simple.csv • full.csv • simple.json • full.json • csv • json For example, **-exportstats** <*local_dir_path*> **-exportstatsformat** *simple.json* You can add multiple arguments separated by a comma. For example, **-exportstats** <*local_dir_path*> **-exportstatsformat** *simple.json*, *full.csv* When you want to export both simple and full type of test results in a json or csv format, you can specify *json* or *csv* as the arguments in the command. When the test run completes, the test result exports to simple.json and full.json files.

Option	Description
	For example, **-exportstats** <*local_dir_path*> **-exportstatsformat** *json* You can select the **Command Line** check box from the product preferences (**Window > Preferences > Test > Performance Test Reports > Export Reports**) when you want to export test results to one of the selected formats after the test run completes. **Remember:** When you run the test from the command line, and if you use the **-exportstats** parameter, then the command line preferences take precedence over the preferences set in the product. Therefore, by default, the test result exports to a CSV format. For example, when you select the **Command Line** option and **Report format** to *json* in the product preferences, and run the test from the command-line interface without using the **-exportstats** option. The result is exported to a json file after the test run is complete.
-exportstatspattern	You can use this option along with the **-exportstats** option to specify a pattern that defines the name of a *csv* or *json* file. You can use the following token to customize the file name: "{TEST_NAME}{AGENT_NAME}{SPLIT_INDEX}_{TIMESTAMP}_{REPORT_ID}_{EXPORT_TYPE}" For example, cmdline -workspace "C:\Users\workspace" -project "proj123" -suite "test1.testsuite -exportstatshtml" C:\Users -exportstatsformat "simple.json,full.csv" -exportstats C:\Users -exportlog "C:\Users\logexport.txt" -exportstatreportlist "http,httpVP" -exportstatspattern "RFT_{REPORT_NAME}_{EXPORT_TYPE}_{REPORT_ID}" **Note:** The **-exportstatspattern** parameter is only supported for the command which is run from the command line of Rational® Functional TesterHCL OneTest™ UI.
-exportstatshtml	When you want to export web analytic results, you can use this option. The results are exported to the specified directory. You can then analyze the results on a web browser without using the test workbench.
-history	Use this option when you want to view a record of all events that occurred during a test run. However, you must use the command suffixed with any of the following options: • jaeger: To send test logs to the Jaeger UI during the *test* run. • testlog: To send test logs as traditional test logs in Rational® Functional TesterHCL OneTest™ UI during the test run. • null: To send no test logs either to the Jaeger UI or Rational® Functional TesterHCL OneTest™ UI during the test run. For example:

Option	Description
	-workspace *workspace_full_path* **-project** *proj_rel_path* **-suite** *suite_rel_path* **-stdout -history** *comma delimited list of modes* **-workspace** *C:/Users/IBM/rationalsdp/test_wsC:/Users/HCL/hclonetest/test_ws* **-project** *Project1* **-suite** *test1.testsuite* **-stdout -history** *jaeger* **Note:** You can add multiple options separated by a comma to send test logs during the test run to Rational® Functional TesterHCL OneTest™ UI and the Jaeger UI. For example: **-workspace** *C:/Users/IBM/rationalsdp/test_wsC:/Users/HCL/hclonetest/test_ws* **-project** *Project1* **-suite** *test1.testsuite* **-stdout -history** *jaeger, testlog*
-import	When you want to run Web UI tests that are in a source control system such as Git from a computer that runs the desktop product, you can clone the project resources in the remote repository to your computer. You can use an empty workspace folder on your computer to import the UI Test project resources and then run the tests from the command-line interface. The desktop product is enabled to run the Web UI tests without the need of the workspace in the cloned repository or your existing workspace. You must use the -workspace command argument to precede the -import command argument. **Note:** You can use this command argument in the following scenarios: • You do not want to use your existing workspace. • You do not want to use the workspace cloned from a remote repository. To run UI tests contained in UI Test projects that are in a remote repository, you must perform the following steps: 1. Clone the remote repository that contains the UI Test project to your computer. 2. Create an empty workspace on your computer. 3. Run the following command from the command-line interface: cmdline -workspace <path_of_empty_workspace> -project <project_name> -import <path_to_-cloned_project_folder> -suite <test_name> For example, cmdline -workspace C:\workspace -project UIProject1 -import d:\work\UIProject1 -suite Test1

Option	Description
	You can also import multiple projects by specifying the project paths separated by a comma. For example, cmdline -workspace C:\workspace -project UIProject1 -import d:\work\UIProject1,d:\work\UIProject2 -suite Test1
-importzip	To import the project as test assets with dependencies into your workspace, use the -importzip option. This command is available from 9.2.1.1 and later.
-labels	You can use the **-labels** option to add labels to test results when you run test assets from the command-line interface. You can add multiple labels to a test result separated by a comma. For example, **-labels** *"label1, label2"* **Note:** If the name of the label contains a space character, then you must enclose it with quotes (*""*). For example, if the name of the label is *test environment*, then you must provide it as `"test environment"`. You can also use the **-labels** option along with the **-publish** option to add labels to a test result when you want to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. When you run test assets from the command-line interface by using the **-labels** option, then the same labels are displayed on the UI Test Statistical Report in Rational® Functional TesterHCL OneTest™ UI. Similarly, when you use the **-labels** option with the **-publish** option from the command-line interface, then the **Results** page of Rational® Test Automation ServerHCL OneTest™ Server displays the same label for the specific test asset.
-overwrite	Determines whether a result file with the same name is overwritten. The default value, `false`, indicates that the new result file is created. If the value is `true`, the file is overwritten and retains the same file name. You must use double quotes *""* for values `true` or `false`.
-plugins	The complete path to the folder that contains the plugins. Typically, on Windows operating systems, this folder is located at `C:\Program Files\IBM\IBMIMShared\pluginsC:\Program Files\HCL\HCLIMShared\plugins`. Required. This option is required only if the folder is at a different location.
- protocolinput	You can use this option with additional arguments as follows:

Option	Description

	• To run a Web UI test in parallel on different browsers
	
	-protocolinput *"all.available.targets.in.parallel=all"*
	
	-protocolinput *"all.available.targets.in.parallel=chrome,ff,ie"*
	
	Note: If you use the **-protocolinput** argument, you must not use the following equivalent **-vmargs** arguments:
	`-vmargs "-Dall.available.targets.in.parallel=all"`
	`-vmargs "-Dall.available.targets.in.parallel=browser1,browswer2,browser3"`
	
	• To specify the Web UI preferences such as highlighting the page element and capturing screens
	
	For example, **-protocolinput** *"webui.highlight=<value>;webui.report.screenshots=<value>"* where *webui.highlight* specifies whether the page element must be highlighted and *webui.report* specifies whether the screens must be captured while playing back the test in the browser.
	
	• To run only the failed tests from a previous playback in an Accelerated Functional Test suite
	
	cmdline -workspace *workspacename* **-project** *projectname* **-aftsuite** *aftsuitname* **-exportlog** *exportlogpath* **-results** *autoresults* **-protocolinput** *"runfailedtests=true"*
	
	In the preceding example, *runfailedtests=true* specifies whether the failed test from a previous playback must be rerun in Accelerated Functional Test suite.
	
	• To automatically resolve the browser and driver incompatibility, while you play back the Web UI tests
	
	-protocolinput *"webui.browser.driver.autoupdate=true"*
	
	• To apply guided-healing and self-healing features while you run Web UI tests
	
	cmdline -workspace *workspacename* **-project** *projectname* **-suite** *test1* **-exportlog** *exportlogpath* **-results** *autoresults* **-protocolinput** *"autoheal=true"*

-publish	You can use **-publish** parameter to publish test results to Rational® Test Automation ServerHCL OneTest™ Server. You can use the following options along with the **-publish** parameter:

Option	Description
	• **no** You can use the **no** option if you do not want to publish test results after the run. This option is useful if the product preferences are set to publish the results, but you do not want to publish them. • You can use any of the following options to specify the project name: ◦ *serverURL* **#project.name**=*projectName***&teamspace.name**=*name_of_the _teamspace* ◦ *serverURL* **#project.name**=*projectName***&teamspace.alias**=*name_of_the _teamspace_alias* You must consider the following points while providing the project name: ◦ If the project name is not specified, then the value of the **-project** parameter is used. ◦ If you have a project with the same name in different team spaces, then you can append either the **&teamspace.name**=*name_of_the _teamspace* or **&teamspace.alias=**=*name_of_-the_teamspace_alias* options along with the **-publish** parameter. For example: -publish “https://localhost:5443/#project.name=test&teamspace.name=ts1” Where: ▪ https://localhost:5443 is the URL of the server. ▪ test is the name of the project. ▪ ts1 is the name of the team space. **Note:** If the name of the project or team space contains a special character, then you must replace it with `%<Hex_value_of_special_character>`. For example, if the name of the team space is *Initial Team Space*, then you must provide it as *Initial%20Team%20Space*. Where, %20 is the hexadecimal value of the space character. **Remember:** If you provide the server and the project details under **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server** in the product and if you use `server-`

Option	Description
	URL#**project.name**=*projectName* along with the **-publish** parameter, the server details in the command-line interface take precedence over the product preferences. **Important:** You must provide the offline user token for the server by using the **RTCP_OFFLINE_-TOKENHCL_ONETEST_OFFLINE_TOKEN** environment variable before you use the **-publish** parameter in the command-line interface.
-publish_for	You can use this option to publish the test results based on the completion status of the tests: • ALL - This is the default option. You can use this option to publish test results for any text execution verdict. • PASS - You can use this option to publish test results for the tests that have passed. • FAIL - You can use this option to publish test results for the tests that have failed. • ERROR - You can use this option to publish test results for the tests that included errors. • INCONCLUSIVE - You can use this option to publish test results for the tests that were inconclusive. You can add multiple parameters separated by a comma.
-publishreports	You can use this option to publish test results in Rational® Test Automation ServerHCL OneTest™ Server. You must use the **-publishreports** parameter along with the **-publish** parameter. You can use the following values: • FT - This is an identifier for Functional Test Report. You can use this value to publish the unified report if it is available for the selected test. See Unified reports on page 1455. • STATS - This is an identifier for Statistics Report. You can use this value to publish the web analytics report if it is available for the selected test. See UI Test Statistical report on page 1469. • TESTLOG - This is an identifier for Test Log. You can use this value to publish the test log if it is available for the selected test. See Logs overview on page 1491. For example, **-publishreports** *"STATS, TESTLOG"* The values specified here override the values selected in **Window > Preferences > Test > Rational Test Automation Server > HCL One Test Server > Results** of Rational® Functional TesterHCL OneTest™ UI. You must prefix with *"!"* to publish all the reports except the specified one. For example, **-publishreports** *"! TESTLOG"* All the reports except the TESTLOG report is published to Rational® Test Automation ServerHCL OneTest™ Server after executing the command.

Option	Description
-quiet	Turns off any message output from the launcher and returns to the command shell when the run or the attempt is complete.
-results	You can use this option to specify the name of the results file. The default result file name is the test name with a time stamp appended. You must specify a folder name that is relative to the project to store the test results. For example, **-results** *folder/resultname*
-stdout	You can use this option to display the information about the test on the command line. After you run a test from the command line, the following outputs are displayed to give you the overall information of the test: • --VERDICT: The verdict of the test. • --REMOTE_RESULT: The URL of the result published to Rational® Test Automation ServerHCL OneTest™ Server. • --REMOTE_RESULT_UI: The URL of the result published to Rational® Test Automation ServerHCL OneTest™ Server and can be opened in a browser to analyze the result. • --LOCAL_RESULT: The path of the result saved locally. For example, **-workspace** *workspace_full_path* **-project** *proj_rel_path* **-publish***publish_url* **-stdout**
-swap-datasets	Use this option to replace dataset values during a test run. If a test is associated with a dataset, you can replace the dataset at run time while initiating the run from the command line. You must ensure that both original and new datasets are in the same workspace and have the same column names. You must also include the path to the dataset when you run the -swapdatasets command. For example, **-swapdatsets** */project_name/ds_path/ds_filename.csv:/project_name/ds_path/new_ds_filename.csv* You can swap multiple datasets that are saved in a different project by adding multiple paths to the dataset separated by a semicolon. For example, `-swapdatsets / project_name1/ds_path/ds_filename.csv:/project_name1/ds_path/new_ds_filename.csv; /project_name2/ds_path/ds_filename.csv:/project_name2/ds_path/new_ds_filename.csv`
-test-Type	You can specify the test type to be selected for test run when you want to run mulitple tests in a workspace by using a single command. You can use the following values: • compound - selects all the compound tests in the project for test run. • test - selects all the Web UI, mobile, HTTP, and Windows tests in the project for test run.

Option	Description
	For example, you can use cmdline -workspace “workspace” -project myProject -testType compound to run all the compound tests in **myProject**.
-user-comments	You can add text within double quotation mark (“”) to display it in the **User Comments** row of the report. **Note:** You can use the file `CommandLine.exe` to run the command to add comments in a language that might not support Unicode characters on Windows operating system.
-varfile	You can use this option to specify the complete path to the XML file that contains the variable name and value pairs. To run a Web UI test on a different browser than that was used for the recording, specify the predefined variable. For more information, see Defining a variable to run a test with a selected browser on page 578.
-vmargs	To specify the Java™ maximum heap size for the Java™ process that controls the command line playback, use the **-vmargs** option with the *-Xmx* argument. For example, when you use**-vmargs** *-Xmx4096m*, specify a maximum heap size of 4096m. This method is similar to specifying *-Xmx4096m* in the eclipse.ini file for the workbench when playing back the test from the user interface. To collect the response time data for the app itself and for the server and network and display them in different bar charts, use **-vmargs** *"-De2e.collect=true"*. For desktop-based web applications, the response time data is captured and displayed by default. To execute tests in parallel on all mobile devices, which are in passive mode, connected to the workbench and ready for playback, use **-vmargs** *"-Dall.available.targets.in.parallel=true"*. To execute tests in parallel on all supported desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=all"*. To execute tests in parallel on selected desktop browsers and connected mobile devices, use **-vmargs** *"-Dall.available.targets.in.parallel=browser1,browswer2,browser3"*. You must separate browser names with a comma. For example, *firefox, ff, chrome, ie, ie64, safari, "-Dall.available.targets.in.parallel=browser1,browser2,browser3"*.
	You can use this option to customize the appearance of the PDF files, before you export the unified reports. The arguments and values are described as follows:

Option	Description
	• To highlight only the verification points in test steps and exclude the tabular form of the step details in the PDF, you must use **-vmargs** *"-Dreport.skipTableInPdf=true"*. To view details of test steps in the tabular form without highlighting the verification points in the PDF, you must use **-vmargs** *"-Dreport.skipTableInPdf=true"*. This setting is the default preference for the exported PDF files for unified reports in IBM® Rational® Functional TesterHCL OneTest™ UI. • To increase the size of screen captures that are displayed in the PDF, you must use **-vmargs** *"-Dreport.snapShotSize=Size of images relative to the page-width"*. You can enter a value between `50` and `100` so that the images are increased to fit within the page margins in the exported PDF. For example, you must enter the following parameter to increase the width of the images to fit within 75% of the page width while the height gets increased proportionately: `-Dreport.snapShotSize=75`  **Note:** ◦ You can use this parameter, only when you have set `-Dreport.skipTableInPdf=true`. ◦ If the parameter value is empty, IBM® Rational® Functional TesterHCL OneTest™ UI automatically sets the size of images to 50% of the page width as the default size. • To exclude screen captures of specific SAP GUI test steps in the PDF, you must use **-vmargs** *"-Dreport.skipTableInPdf=objectType1:methodName1|objectType2:methodName2"*. You can use the pipe character `|` as the delimiter between different objects. For example, you must enter the following values for the parameter to exclude images captured for the `maximize` and `setTest` methods of the `GuiMainWindow` and `GuiCheckBox` objects, respectively: `-Dreport.skipActionEventImageList=GuiMainWindow:maximize|GuiCheckBox:setTest`  **Note:** ◦ This parameter is applicable for SAP GUI applications only. ◦ You can use this parameter only when you set `-Dreport.skipTableInPdf=true`. ◦ You can use this parameter only if you have installed IBM® Rational® Performance TesterHCL OneTest™ Performance with IBM® Rational® Functional TesterHCL OneTest™ UI in the shell-shared mode. For more information, see Customizing PDF files exported from unified reports on page 1464.

Running a test from the command line

You can run Web UI, mobile, and windows tests using the command-line interface of Rational® Functional TesterHCL OneTest™ UI.

Before you begin

You must have completed the following tasks:

- Recorded the tests to be run.
- Created a config file and varfile as required for tests. See Configuration of test runs from the command line on page 1381.

1. Go to one of the following directories:
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.sh` on Linux operating system.
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.sh` on Mac operating system.
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.bat` on Windows operating system.
2. Close Rational® Functional TesterHCL OneTest™ UI, if it is open, before running the cmdline command.
3. Enter the following command to run the test:
   cmdline -workspace <workspacename> -project <projectname> -suite <suitename>

   For example:

   ```
   cmdline -workspace D:\My Workspace -project myProject -suite Tests\myWebUITest.testsuite
   ```

   The test execution starts and the status is displayed on the screen.

   Note:
   - The workspace is locked after you issue the command. To check the progress of the test during the run, invoke another workspace and open the project through that workspace.
   - On Linux operating system, the command must start with cmdline.sh.

   The command line syntax with the supported options is as follows:

   cmdline -workspace <workspace_full_path> -project <proj_relative_path> -eclipsehome <eclipse_full_path> -plugins <plugin_full_path> -suite <suite_relative_path> -importzip <full_path.zip> -varfile <variable_file_full_path> -configfile <file_full_path> -results <result_file> -overwrite <{"true" | "false"}> -quiet -vmargs <JVM_args> -publish <serverURL#project.name=projectName -publish_for {ALL,PASS,FAIL,ERROR, INCONCLUSIVE}> -labels <labelname1, labelname2> -exportlog <log_full_path> -exportstats <local_dir_path> -exportstatshtml <local_dir_path> -exportstatsformat <name of the file format> -compare <"result_path1, result_path2"> -exportstatreportlist <stats_list> -execsummary <local_dir_path> -execsummaryreport

<reportID> -usercomments <"any user comment"> -publishreports <"FT, STATS, TESTLOG"> -stdout -swapdatsets <existing_dataset_file_path:new_dataset_file-path>

If a value contains spaces, enclose the value in quotation marks. To see the online help for this command while you are in the directory that contains the `.bat` file, type cmdline -help.

To stop the test run, you can open another command prompt window and use one of the following options with the cmdline option:

Command	Description
-stoprun	Optional. Stops the test run after the specified number of seconds. The block is executed, and the test log is transferred before stopping the run. You must use the **-workspace** command and specify the location of the workspace.
-abandon-run	Optional. Stops the test run immediately. You must use the **-workspace** command and specify the location of the workspace.

Note: Messages are displayed to indicate when the test is launched and when it is completed unless you include the -quiet option.

What to do next

After you run the test you may want to export the results for further analysis. For more information, see Exporting report counters automatically.

Running multiple tests from the command line

When you want to run multiple tests or compound tests that are in a project folder, you can select and run those tests by using any of the command options such as "*" or testType from the command line.

Before you begin

You must have completed the following tasks:

- Recorded multiple tests or created multiple compound tests that are contained in your project folder.
- Read and been familiar with the command-line arguments to use with the cmdline command. See Configuration of test runs from the command line on page 1381.

1. Navigate to the directory that contains the `cmdline.bat` or `cmdline.sh` file.
2. Enter one the following commands to run multiple tests:

   cmdline -workspace <workspacename> -project <projectname> -suite "*"

cmdline -workspace <workspacename> -project <projectname> -testType <testtype_value>

Where,

- “*” is the wildcard character enclosed within double quotes that denotes all test assets in the specified project folder.
- <testtype_value> indicates the test type to be selected for test run. Rational® Functional TesterHCL OneTest™ UI supports **test** and **compound** as <testtype_value>. You can specify the value as **test** for all individual tests such as Web UI, mobile, HTTP, and Windows tests. For all compound tests, you can specify the value as **compound**.

You can also use the exclude option “!” along with the wildcard option “*” to exclude the specified test assets.

The following table lists the sample commands to run multiple tests.

Option	**Sample command**	**Description**
Wildcard character '*'	cmdline -workspace “workspace” -project myProject -suite "*"	This command runs all the tests in **myProject**.
	cmdline -workspace “workspace” -project myProject -suite "test*"	This command runs all the tests in **myProject** that have the file names starting with 'test'.
	cmdline -workspace “workspace” -project myProject -suite "Compound Tests/*"	This command runs all the tests in the **Compound Tests** folder under **myProject**.
	cmdline -workspace “workspace” -project myProject -suite "Web*.testsuite:/myProject2/Test1*.testsuite"	This command runs all the tests in **myProject** that have the file names starting with 'Web' and all the tests in **myProject2** that have the file names starting with 'Test1'.
	cmdline -workspace "workspace" -project "UI1" -aftsuite "AFT*"	This command runs all the AFT tests that have file names starting with 'AFT'.
Wildcard character '*' with '!' to exclude a specified value	cmdline -workspace “workspace” -project myProject -suite "Web*.testsuite:!Web2"	This command runs all the tests in **myProject** that have the file names starting with 'Web', but not 'Web2'.

Option	Sample command	Description
testType	cmdline -workspace “workspace” -project myProject -testType compound	This command runs all the compound tests in **myProject**.
	cmdline -workspace “workspace” -project myProject -testType test	This command runs all the Web UI, mobile, HTTP, and Windows tests in **myProject**.

Results

You have run multiple tests that are contained in your project folder by using the wildcard or testType option from the command line.

What to do next

After you run the test, you can verify results in the unified report. See Unified reports for Web UI, mobile, and Windows tests on page 1456.

Creating a command-line config file

Starting from 10.0.2, you can create command line config file from the product, which you can use while running tests from the command-line interface and Maven.

Before you begin

You must have performed the following tasks:

- Created test assets in a workspace.
- Installed Maven if you are running tests from the Maven build.

For information about creating tests and installing Maven, see related links.

About this task

Previously, you created the config file manually by adding parameters to it for running the tests by using the config file from the command line. Now, you can create a command-line config file from the product by right-clicking the test asset. The required parameters are automatically assigned, and you can specify any optional parameters, while creating the config file. You can use this config file to run the tests from the command-line interface and Maven plug-in that is provided with the product as part of Maven build.

1. In the Test Navigator, browse and select the test.
2. Right-click the test, and then click **Create command line config file**.
3. In the **Create New Config File** window, enter a name for the new configuration file and then click **Next**.
4. Perform the following sub-steps in the **Command Line Arguments** window:

a. Select the format of the config file from the following options:
   - **Regular** – Use this format to run tests from the command-line interface.
   - **Maven** – Use this format to run tests from the Maven build.

b. If you want to add more parameters to a config file, specify the values in the fields from the available configuration options.

5. Click **Finish**.

Results

The **Config file created** dialog box displays the location of the config file.

What to do next

You must complete the following steps:

1. Close the product.
2. Run a test by using the config file either from the command-line interface or from the Maven build.

Related information

Creating Web UI tests on page 532

Testing with Maven on page 393

Running Web UI tests from the command-line

You can run a Web UI test without using the desktop client by using the command-line interface. You must use the **-varfile** parameter that specifies the complete path to the XML file in the command. The XML file contains the name-value pairs of the variables. The variables file specifies the web browser on a computer, device, emulator, or device on a cloud on which to run the test.

Running the command

Before you can run the command to run Web UI tests from the command line, you must create a variable file that contains the details of the computer, device, emulator, or device on a cloud on which you want to specify the web browser to run the test. You must have also recorded the Web UI test.

You can issue the following command to run a Web UI test from the command-line:

cmdline -workspace <workspace_full_path> -project <proj_relative_path> -eclipsehome <eclipse_full_path> -plugins <plugin_full_path> -suite <suite_relative_path> -importzip <full_path.zip> -varfile <variable_file_full_path> -configfile <file_full_path> -results <result_file> -overwrite <{"true" | "false"}> -quiet -vmargs <JVM_args> -publish <serverURL#project.name=projectName -publish_for {ALL,PASS,FAIL,ERROR, INCONCLUSIVE}> -labels <labelname1, labelname2> -exportlog <log_full_path> -exportstats <local_dir_path> -exportstatshtml <local_dir_path> -exportstatsformat <name of the file format> -compare <"result_path1, result_path2"> -exportstatreportlist

<stats_list> -execsummary <local_dir_path> -execsummaryreport <reportID> -usercomments <"any user comment"> -publishreports <"FT, STATS, TESTLOG"> -stdout -swapdatsets <existing_dataset_file_path:new_dataset_file-path>

Important: When you use the command to run a Web UI test, you must ensure that the following conditions are satisfied:

- Use the variable file that contains the details of the computer, device, emulator, or device on a cloud on which you want to select the web browser.
- Specify the details of the variable file in the command.

Creating a variable file

You must create a variable file that contains the variable names and values as pairs that are required to connect to the web browser. You can connect and run the Web UI tests on browsers on any of the following computers, devices, emulators, or devices on a cloud:

- Computer on which you have installed Rational® Functional TesterHCL OneTest™ UI on page 1402
- UI Test Agent on page 1402
- BitBar Cloud on page 1403
- BrowserStack Cloud on page 1404
- Perfecto mobile Cloud on page 1405
- pCloudy Cloud on page 1406

Variable file for a computer on which you have installed Rational® Functional TesterHCL OneTest™ UI

You can create an XML file as a variable file that specifies the browser to use on the computer on which you have installed Rational® Functional TesterHCL OneTest™ UI. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<browser>" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

For example, if you connect *Emulator:Pixel_2_API_28* as the mobile emulator and you want to use the Chrome browser to run the test, the variable file can be as follows:

```
<?xml version="1.0" encoding="UTF-8"?><inits>
<variable_init value="Chrome(Emulator:Pixel_2_API_28)" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

Variable file for UI Test Agent

You can create an XML file as a variable file that specifies the browser to use on a device or an emulator connected to the remote computer. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<browser(device or emulator name)>" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="<UI Test Agent host URL>" name="appium.server.host"/>
<variable_init value="<port number>" name="appium.server.port"/>
</inits>
```

The following table lists the variables and the actions required for the value field:

Variable	Action
RTW_WebUI_Browser_Selection	Specify the browser to be used on the device or emulator that is connected to the UI Test Agent on the remote computer.
appium.server.host	Specify the host name or IP address of the remote computer that contains the UI Test Agent. **Note:** The default value for this variable is 127.0.0.1. If no value is specified, the default value is used during the playback.
appium.server.port	Specify the port number of the UI Test Agent that is installed on the remote computer.

For example, if you connect *Emulator:Pixel_2_API_28* as the mobile emulator to a remote computer where the UI Test Agent URL is *10.115.50.61*, and the UI Test Agent port is *7082*, and you want to use the Chrome browser, the variable file can be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="Chrome(Emulator:Pixel_API_28)" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="10.115.50.61" name="appium.server.host"/>
<variable_init value="7082" name="appium.server.port"/>
</inits>
```

Variable file for BitBar Cloud

You can create an XML file as a variable file that specifies the browser to be used on the device on the BitBar Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<Browser(BitBar device name)>" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="<BitBar API key>" name="bitbar.apikey"/>
<variable_init value="<BitBar host URL>" name="bitbar.host"/>
<variable_init value="<project name>" name="bitbar.project"/>
<variable_init value="<test name>" name="bitbar.testrun"/>
</inits>
```

The following table lists the variables and the sample values:

Variable	Action	Sample value
RTW_WebUI_Browser_Selection	Specify the browser to be used on the mobile device that is connected to the BitBar Cloud.	Chrome(BitBar:Google Pixel 2)
bitbar.apikey	Specify the user token generated for your BitBar account to authenticate your connection with the BitBar Cloud.	LkBldnjcnzrIcwWZpCZZxy
bitbar.host	Specify the host name of the BitBar Cloud instance.	appium.bitbar.com
bitbar.project	Specify the name of the project that contains the recorded test.	PlaybackWebUI
bitbar.testrun	Specify a name for the test run that must be displayed in the BitBar dashboard for the test run.	CLIExecution

The example variable file for the values in the table is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="Chrome(BitBar:Google Pixel 2)" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="LkBldnjcnzrIcwWZpCZZxy" name="bitbar.apikey"/>
<variable_init value="appium.bitbar.com" name="bitbar.host"/>
<variable_init value="PlaybackWebUI" name="bitbar.project"/>
<variable_init value="CLIExecution" name="bitbar.testrun"/>
</inits>
```

Variable file for BrowserStack Cloud

You can create an XML file as a variable file that specifies the browser to be used on the device on the BrowserStack Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BrowserStack user name>" name="browserstack.username"/>
<variable_init value="<BrowserStack host URL>" name="browserstack.host"/>
<variable_init value="<BrowserStack API key>" name="browserstack.apikey"/>
<variable_init value="<project name>" name="browserstack.project"/>
<variable_init value="<Browser(BrowserStack : device name)>" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

The following table lists the variables and the sample values:

Variable	Action	Sample value
browserstack.username	Specify the user name of your BrowserStack account to authenticate your connection with the BrowserStack Cloud.	tester_1

Variable	Action	Sample value
browserstack.host	Specify the host name of the BrowserStack Cloud instance.	api-cloud.browserstack.com
browserstack.apikey	Specify the user token generated for your BrowserStack account to authenticate your connection with the BrowserStack Cloud.	KBk7PjhpqgfeVmxasUye
browserstack.project	Specify the name of the project that contains the recorded test.	CMDLINETest
RTW_WebUI_Browser_Selection	Specify the browser to be used on the mobile device that is connected to the BrowserStack Cloud.	Chrome(BrowserStack : Google Pixel 7 Pro)

The example variable file for the values in the table is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="tester_1" name="browserstack.username"/>
<variable_init value="api-cloud.browserstack.com" name="browserstack.host"/>
<variable_init value="KBk7PjhpqgfeVmxasUye" name="browserstack.apikey"/>
<variable_init value="CMDLINETest" name="browserstack.project"/>
<variable_init value="Chrome(BrowserStack : Google Pixel 7 Pro)" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

If you want to run a compound test that contains a Web UI test and a mobile test, then you can specify the device and browser by using *Mobile_Device_Selection* and *RTW_WebUI_Browser_Selection*. For example,

```
<variable_init value="BrowserStack : Google Pixel 7 Pro" name="Mobile_Device_Selection"/>
<variable_init value="Chrome(BrowserStack : Google Pixel 7 Pro)" name="RTW_WebUI_Browser_Selection"/>
```

Variable file for Perfecto Cloud

You can create an XML file as a variable file that specifies the browser to be used on the device on the Perfecto Mobile Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<browser(device name)>" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="<security token>" name="perfecto.securitytoken"/>
<variable_init value="<perfecto host URL>" name="perfecto.host"/>
</inits>
```

The following table lists the variables and the sample values:

Variable	Action	Sample value
RTW_WebUI_Brows-	Specify the browser to be used on the mobile device that is connected to the Perfecto Cloud.	Chrome(Perfecto:R48904TNSAZ)

Variable	Action	Sample value
er_Selection		
perfecto.security-token	Specify the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.	LkBldnjcnzrIcwWZpCZZxy
perfecto.host	Specify the URL of the Perfecto Cloud.	partners.perfectomobile.com

The example variable file for the values in the table is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="Chrome(Perfecto:R48904TNSAZ)" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="LkBldnjcnzrIcwWZpCZZxy" name="perfecto.securitytoken"/>
<variable_init value="partners.perfectomobile.com" name="perfecto.host"/>
</inits>
```

Variable file for pCloudy Cloud

You can create an XML file as a variable file that specifies the browser on the device connected to the pCloudy Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<browser(device name)>" name="RTW_WebUI_Browser_Selection"/>
<variable_init value="<api key>" name="pcloudy.apikey"/>
<variable_init value="<pcloudy host URL>" name="pcloudy.host"/>
<variable_init value="<pcloudy user name>" name="pcloudy.username"/>
</inits>
```

The following table lists the variables and the sample values:

Variable	Action	Sample value
RTW_WebUI_Browser_Selection	Specify the browser to be used on the device that is connected to the pCloudy Cloud.	*Chrome(pCloudy:SAMSUNG_GalaxyS8_Android_-7.0.0_3ced4*)
pcloudy.apikey	Specify the API key of your pCloudy account to authenticate the user name for a successful connection.	*csv7wxjyyzzgrzbqym62cvfg*
pcloudy.host	Specify the URL of the pCloudy Cloud instance.	*device.pcloudy.com*
pcloudy.username	Specify the user name of your pCloudy account.	*myUserName@pcloudy*

The example variable file for the values in the table is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="Chrome(pCloudy:SAMSUNG_GalaxyS8_Android_7.0.0_3ced4)"
 name="RTW_WebUI_Browser_Selection"/>
<variable_init value="csv7wxjyyzzgrzbqym62cvfg" name="pcloudy.apikey"/>
<variable_init value="device.pcloudy.com" name="pcloudy.host"/>
<variable_init value="myUserName@pcloudy" name="pcloudy.username"/>
</inits>
```

Related information

Configuration of test runs from the command line on page 1381

Running a Web UI test or compound test in the headless mode from the command line

When you do not want to view the GUI as the recorded test is played back on a web browser, you can choose to run the test in the headless mode from the command line. Running of tests in the headless mode is useful when you use an automated testing environment.

Before you begin

You must have completed the following tasks:

- You must have recorded a single Web UI test, compound test with multiple Web UI tests, or an AFT Suite that you want to run in the headless mode.
- Read and been familiar with the command line arguments to use with the cmdline command. See Configuration of test runs from the command line on page 1381.

1. Navigate to the directory that contains the `cmdline.bat` or `cmdline.sh` file.

   On Windows™ operating systems, this directory can be as *`productInstallationDirectory`*`\cmdline`. For example, `C:\Program Files\IBM\RFT\cmdlineC:\Program Files\HCL\HCLProducts\UI \cmdline`.

2. Issue the following argument for the specific browser along with the other arguments in the cmdline command to run the test in the headless mode:

Browser	Argument
Chrome	-protocolinput "webui.chrome.headlessmode.selected=TRUE"
Edge	-protocolinput "webui.edge.headlessmode.selected=TRUE"
Firefox	-protocolinput "webui.firefox.headlessmode.selected=TRUE"

   For example, consider the following values for the different arguments:

Argument	Value
-workspace	*D:\My Workspace*
-project	*myProject*
-eclipsehome	*C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest*
-plugins	*C:\Program Files\IBM\IBMIMShared\pluginsC:\Program Files\HCL\HCLIMShared \plugins*
-suite	*myWebUITest.testsuite*

If you want to run the Web UI test or compound test in the headless mode by using the Chrome browser, then the arguments that you can enter in the command line are as follows:

```
cmdline -workspace D:\My Workspace -project myProject -eclipsehome C:\Program Files\IBM\SDP
 -plugins C:\Program Files\IBM\IBMIMShared\plugins -suite Tests\myWebUITest.testsuite
 -protocolinput "webui.chrome.headlessmode.selected=TRUE"
```

```
cmdline -workspace D:\My Workspace -project myProject -eclipsehome C:\Program
 Files\HCL\HCLOneTest -plugins C:\Program Files\HCL\HCLIMShared\plugins -suite
 Tests\myWebUITest.testsuite -protocolinput "webui.chrome.headlessmode.selected=TRUE"
```

Related information

Running tests from a schedule on page 1363

Running a Web UI test using IBM Rational Quality Manager on page 1361

Running Web UI tests in the private mode from the command line

When you play back Web UI tests, if the user profile that is stored in the browser prevents the test from running without intervention, then you can use the private or incognito mode from the command line. Also, when you do not want cookies and caches to affect your test play back, you can choose the private mode.

Before you begin

You must have completed the following tasks:

- Recorded a Web UI test to run in the private mode.
- Read and been familiar with the command line arguments to use with the cmdline command. See Configuration of test runs from the command line on page 1381.
- Read and been familiar with the usage of variable files. See Using a variable file on page 1382.

1. Create a variable file with the following variable name and its value:
   <variable_init name="<browser_name>.private" value="true" />

   Use the following values for the specific browser:

Browser	Value
Chrome	<variable_init name="chrome.private" value="true" />
Edge	<variable_init name="edge.private" value="true" />
Firefox	<variable_init name="firefox.private" value="true" />
Opera	<variable_init name="opera.private" value="true" />

The sample variable file for running a test in the private mode of Chrome browser is as follows:

```
<?xml version="1.1" encoding="UTF-8"?>
<inits xmlns="http://www.ibm.com/rational/test/lt/varinit">
    <variable_init name="chrome.private"
        value="true" />
</inits>
```

2. Go to one of the following directories:
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.sh` on the Linux operating system.
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.sh` on the Mac operating system.
   - `<productInstallationDirectory>\cmdline` directory that contains `cmdline.bat` on the Windows operating system.
3. Close Rational® Functional TesterHCL OneTest™ UI, if it is open, before you run the cmdline command.
4. Enter the following command to run the test in the private mode:
   cmdline -workspace <workspacename> -project <projectname> -suite <suitename> -varfile <variable_file_full_path>

   For example:

   ```
   cmdline -workspace D:\My Workspace -project myProject -suite Tests\myWebUITest.testsuite -varfile
    "D:\My Workspace\priv.varinit"
   ```

   The test run starts and the status is displayed on the screen.

Results

You have run the test in the private mode from the command line.

Running tests by using the browser launch arguments from the command line

You can open the browser with specific browser launch arguments when you play back Web UI tests through the command line. You can specify the required arguments through a variable file.

Before you begin

You must have completed the following tasks:

- Recorded a Web UI test. See Recording a Web UI test on page 545.
- Read and been familiar with the command line arguments to use with the cmdline command. See Configuration of test runs from the command line on page 1381.
- Read and been familiar with the usage of variable files. See Using a variable file on page 1382.

1. Create a variable file with the following variable name and value pairs:
   <variable_init name="webui.browserparam.selected" value="true"/> <variable_init name="browser.parameters" value="-<param1>;-<param2>" />

   The format of the variable file for passing arguments to browsers is as follows:

   ```
   <?xml version="1.1" encoding="UTF-8"?>
   <inits xmlns="http://www.ibm.com/rational/test/lt/varinit">
   <variable_init name="webui.browserparam.selected" value="true"/>
   <variable_init name="browser.parameters" value="-<param1>;-<param2>" />
   </inits>
   ```

   The sample variable file for passing arguments to browsers is as follows:

   ```
   <?xml version="1.1" encoding="UTF-8"?>
   <inits xmlns="http://www.ibm.com/rational/test/lt/varinit">
   <variable_init name="RTW_WebUI_Browser_Selection" value="firefox"/>
   <variable_init name="webui.browserparam.selected" value="true"/>
   <variable_init name="browser.parameters" value="-private;-headless" />
   </inits>
   ```

2. Go to one of the following directories:
   - `<Install_Directory>\cmdline` directory that contains `cmdline.sh` on the Linux operating system.
   - `<Install_Directory>\cmdline` directory that contains `cmdline.sh` on the Mac operating system.
   - `<Install_Directory>\cmdline` directory that contains `cmdline.bat` on the Windows operating system.
3. Close Rational® Functional TesterHCL OneTest™ UI, if it is open, before you run the cmdline command.
4. Enter the following command to run the test with the specified arguments:
   cmdline -workspace <workspacename> -project <projectname> -suite <suitename> -varfile <variable_file_full_path>

   For example:

   ```
   cmdline -workspace D:\My Workspace -project myProject -suite Tests\myWebUITest.testsuite -varfile
    "D:\My Workspace\args.varinit"
   ```

   The test run starts and the status is displayed on the screen.

Results

You have run the test in the browser with browser launch arguments from the command line.

Running a Web UI test or compound test from the command line on multiple browsers

In addition to running a Web UI test from the product interface, you can automate the test effort by running the test from the command line. To accelerate test execution, you can run a single Web UI test and/or a compound test containing Web UI tests on multiple browsers and devices simultaneously.

About this task

Use the `-vmargs` command line argument to run the Web UI or compound test on multiple browsers or on all browsers and connected mobile devices simultaneously. Use variable names from the following table to specify the browsers to run the command on. To specify multiple browsers, separate the variable names with commas, for example, *ff,chrome,ie*. To run the test on all browsers and connected mobile devices, use the variable name *all*. Do not use spaces for the `-vmargs` arguments.

Browser	Variable
Mozilla Firefox	ff
Google Chrome	chrome
Internet Explorer	ie
Microsoft Edge	edge
Apple Safari	safari

1. To run a test from the command line, go to the directory that contains the `cmdline.bat` and `cmdline.sh` files. On Windows™ operating systems, this directory is typically *`productInstallationDirectory`*`/cmdline`, for example, `C:\Program Files\__BRAND_NAME__\__SDP_PATH__\cmdline`.
2. Issue the `cmdline` command, followed by the arguments defined in the Configuration of test runs from the command line on page 1381 topic.
3. To run the test simultaneously on all supported desktop browsers and connected mobile devices, use the `-vmargs` argument, as follows:

   ```
   -vmargs "-Dall.available.targets.in.parallel=all"
   ```

4. To run the test simultaneously on a selected set of browsers, use the `-vmargs` argument, as follows:

   ```
   -vmargs "-Dall.available.targets.in.parallel=ie,ff,chrome"
   ```

Exemple

```
cmdline -workspace D:\My Workspace -project myProject -eclipsehome C:\Program Files\IBM\SDP
 -plugins C:\Program Files\IBM\IBMIMShared\plugins -suite Tests\myWebUITest.testsuite -vmargs
 "-Dall.available.targets.in.parallel=ie,ff,chrome"
```

```
cmdline -workspace D:\My Workspace -project myProject -eclipsehome C:\Program Files\HCL\HCLOneTest
 -plugins C:\Program Files\HCL\HCLIMShared\plugins -suite Tests\myWebUITest.testsuite -vmargs
 "-Dall.available.targets.in.parallel=ie,ff,chrome"
```

Related information

Running tests from a schedule on page 1363

Running a Web UI test using IBM Rational Quality Manager on page 1361

Using an XML file to run multiple Web UI tests and compound tests simultaneously from the command line

You can use an XML file and the command line to run multiple Web UI tests and compound tests simultaneously on instances of Chrome and Firefox. You can also distribute these tests across multiple remote agent computers. Starting from 9.1.1.1, this feature is also available from the command line.

Before you begin

- Read and been familiar with the command line arguments to use with the cmdline command. See Configuration of test runs from the command line on page 1381.
- There are licensing limitations when attempting to run multiple combinations of tests, browsers, and remote agents. See License descriptions on page 302.
- See Running multiple Web UI tests on multiple browsers and platforms simultaneously on page 1354 for the installation and shell-sharing requirements for running tests across multiple browsers and remote agents.

1. Set up an XML file similar to one of the following samples.

   Sample XML file 1

   This sample XML file lists the full path to the tests and compound tests to run and the browsers (Firefox, Chrome, or both) on which to run the tests. An XML file such as this one is supported in 9.1 and later.

```
<?xml version="1.0" encoding="UTF-8"?>
  <inits>
    <tests>
      <test path="/TestProject/CompoundTests/ACompound.testsuite"/>
      <test path="/TestProject/Tests/AWebTest.testsuite"/>
    </tests>
    <browsers>
      <browser name="chrome"/>
      <browser name="firefox"/>
    </browsers>
  </inits>
```

   Sample XML file 2

   Starting from 9.1.1.1, you can also list the remote agents where you can run the tests. This sample XML file automatically distributes multiple tests across different browsers, remote agents, and the local computer.

   In this sample, tests that run in Chrome use Chrome Device Mode to emulate an Apple iPhone6 Plus and a Google Nexus 5 and also run in Chrome headless mode. With headless mode, tests can run in an automated testing environment where a visible user interface shell is not required.

```
<?xml version="1.0" encoding="UTF-8"?>
  <inits>
    <group>
      <tests>
        <test path="/WebUProj/ariesweb1.testsuite"/>
        <test path="/WebUProj/ariesweb2.testsuite"/>
        <test path="/WebUProj/ariesweb3.testsuite"/>
        <test path="/WebUProj/ariesweb4.testsuite"/>
        <test path="/WebUProj/ariesweb5.testsuite"/>
      </tests>
      <browsers>
        <browser name="chrome" devicemode="Apple iPhone 6 Plus" headless="true"/>
        <browser name="chrome" devicemode="Google Nexus 5"/>
        <browser name="firefox"/>
      </browsers>
      <locations>
        <location      host="9.113.29.29"/>
        <location      host="9.113.29.30"/>
        <location      host="9.113.29.31"/>
        <location      host="9.113.29.32"/>
        <location      host="civcez228.company1.com"/>
      </locations>
    </group>
    <group>
      <tests>
        <test path="/WebUProj/ariesweb6.testsuite"/>
      </tests>
      <browsers>
  <browser name="chrome" devicemode="Apple iPhone6 Plus" headless="true"/>
  <browser name="firefox"/>
      </browsers>
      <locations>
  <location host="localhost"/>
      </locations>
   </group>
  </inits>
```

2. If you are running tests on emulators and device clouds, edit the XML file to add the following values:

```
      <browsers>
        <browser name="<name_of_the_browser>" id="Emulator: <device_name>"/>
        <browser name="<name_of_the_browser>" id="<device_cloud_name>: <device_name>"/>
       </browsers>
```

   For example,

```
      <browsers>
        <browser name="Chrome" id="Emulator:Google_Pixel_7_Pro"/>
        <browser name="Chrome" id="BrowserStack : Google Pixel 7 Pro"/>
      </browsers>
```

3. If you are running a compound test that contains Web UI tests and mobile tests by using a .varinit file, edit the XML file to add the values for *Mobile_Device_Selection* and *RTW_WebUI_Browser_Selection*:

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

```
   <variable_init value="BrowserStack user name" name="browserstack.username"/>
   <variable_init value="BrowserStack host URL" name="browserstack.host"/>
   <variable_init value="BrowserStack API key" name="browserstack.apikey"/>
   <variable_init value="project name" name="browserstack.project"/>
   <variable_init value="BrowserStack : device name" name="Mobile_Device_Selection"/>
   <variable_init value="Browser(BrowserStack : device name)"
name="RTW_WebUI_Browser_Selection"/>
  </inits>
```

For example,

```
<?xml version="1.0" encoding="UTF-8"?>
  <inits>
   <variable_init value="tester_1" name="browserstack.username"/>
   <variable_init value="api-cloud.browserstack.com" name="browserstack.host"/>
   <variable_init value="KBk7PjhpqgfeVmxasUye" name="browserstack.apikey"/>
   <variable_init value="CMDLINETest" name="browserstack.project"/>
   <variable_init value="BrowserStack : Google Pixel 7 Pro" name="Mobile_Device_Selection"/>
   <variable_init value="Chrome(BrowserStack : Google Pixel 7 Pro)"
name="RTW_WebUI_Browser_Selection"/>
  </inits>
```

4. Issue the `cmdline` command as shown in the following example for a Windows™ computer. The XML file is specified in the -aftsuite argument for an AFT Suite or the varfile argument for the XML file with the `.varinit` extension.

Exemple

AFT Suite:

cmdline -workspace C:\workspaces\workspace -project Demo_Proj -plugins "C:\Program Files\HCL\HCLIMShared \plugins" -eclipsehome "C:\Program Files\HCL\HCLOneTest" -aftsuite "aftSuiteName" -results "Results \webUItest_on_off" -exportlog "C:\temp\webLog.txt"

Variable file:

cmdline -workspace C:\workspaces\workspace -project Demo_Proj -plugins "C:\Program Files\HCL\HCLIMShared \plugins" -eclipsehome "C:\Program Files\HCL\HCLOneTest" -suite Tests\myWebUITest.testsuite -varfile "C:\temp \comp.varinit"

AFT Suite:

cmdline -workspace C:\workspaces\workspace -project Demo_Proj -plugins "C:\Program Files\IBM\IBMIMShared \plugins" -eclipsehome "C:\Program Files\IBM\SDP" -aftsuite "aftSuiteName" -results "Results\webUItest_on_off" -exportlog "C:\temp\webLog.txt"

Variable file:

cmdline -workspace C:\workspaces\workspace -project Comp_Proj -plugins "C:\Program Files\IBM\IBMIMShared \plugins" -eclipsehome "C:\Program Files\IBM\SDP" -suite Tests\myWebUITest.testsuite -varfile "C:\temp \comp.varinit"

Related information

Running a Web UI test or compound test from the command line on multiple browsers on page 1411

Running a Web UI test or compound test in the headless mode from the command line on page 1407

Running Web UI tests in the private mode from the command line on page 1408

Running mobile tests for Android applications from the command-line

You can run a mobile test without using the desktop client by using the command-line interface. You must use the **-varfile** parameter that specifies the complete path to the XML file in the command. The XML file contains the name-value pairs of the variables. The variables specify the path to the computer to which the Android device or emulator is connected, and the other configurations required to run the mobile test on the connected device.

Running the command

Before you can run the command to run mobile tests from the command line, you must create a variable file that contains the details of the computer to which the mobile device is connected. You must have also recorded the mobile test.

You can issue the following command to run a mobile test from the command-line:

cmdline -workspace <workspace_full_path> -project <proj_relative_path> -eclipsehome <eclipse_full_path> -plugins <plugin_full_path> -suite <suite_relative_path> -importzip <full_path.zip> -varfile <variable_file_full_path> -configfile <file_full_path> -results <result_file> -overwrite <{"true" | "false"}> -quiet -vmargs <JVM_args> -publish <serverURL#project.name=projectName -publish_for {ALL,PASS,FAIL,ERROR, INCONCLUSIVE}> -labels <labelname1, labelname2> -exportlog <log_full_path> -exportstats <local_dir_path> -exportstatshtml <local_dir_path> -exportstatsformat <name of the file format> -compare <"result_path1, result_path2"> -exportstatreportlist <stats_list> -execsummary <local_dir_path> -execsummaryreport <reportID> -usercomments <"any user comment"> -publishreports <"FT, STATS, TESTLOG"> -stdout -swapdatsets <existing_dataset_file_path:new_dataset_file-path>

Important: When you use the command to run a mobile test, you must ensure that the following conditions are satisfied:

- Use the variable file that contains the details of the computer, server, or cloud to which the Android devices or emulators are connected.
- Specify the details of the variable file in the command.

Creating a variable file

You can create a variable file that contains the variable names and values as pairs that are required to connect to the Android device or emulator. You can connect and run the mobile tests on Android devices or emulators that are connected to the following computers, servers, or mobile clouds:

- Computer on which you have installed Rational® Functional TesterHCL OneTest™ UI on page 1416
- UI Test Agent on page 1417
- BitBar Cloud on page 1418
- BrowserStack Cloud on page 1419
- Perfecto Mobile Cloud on page 1420
- pCloudy Cloud on page 1422
- Variable file to run mobile tests on multiple mobile clouds on page 1423

Variable file for a computer on which you have installed Rational® Functional TesterHCL OneTest™ UI

You can create a variable file in XML format to specify the details required to run the mobile test on selected device.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<Android device or emulator name>" name="Mobile_Device_Selection"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the Android device or emulator that is connected to the computer.	Emulator:Pixel_2_API_28
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for UI Test Agent

You can create an XML file as a variable file that specifies the details of the remote computer on which you have installed the UI Test Agent and connected the Android device or an emulator.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<Android device or emulator name>" name="Mobile_Device_Selection"/>
<variable_init value="<UI Test Agent host URL>" name="appium.server.host"/>
<variable_init value="<port number>" name="appium.server.port"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the Android device or emulator that is connected to the computer.	Emulator:Pixel_2_API_28
appium.server.host	Specify the host name or IP address of the remote computer on which you have installed the UI Test Agent. **Note:** The default value for this variable is 127.0.0.1. If no value is specified, the default value is used during the playback.	10.115.50.61
appium.server.port	Specify the port number of the UI Test Agent that is installed on the remote machine.	7082
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true

Variable	Action	Sample value
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for BitBar Cloud

You can create an XML file as a variable file that specifies the details of the BitBar Cloud.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices and applications.

Restriction: Resource Monitoring metrics of mobile device resources are not collected when the tests are run as an AFT suite.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BitBar device name>" name="Mobile_Device_Selection"/>
<variable_init value="<BitBar API key>" name="bitbar.apikey"/>
<variable_init value="<BitBar host URL>" name="bitbar.host"/>
<variable_init value="<project name>" name="bitbar.project"/>
<variable_init value="<test name>" name="bitbar.testrun"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to BitBar Cloud, create the XML file without the variable *Mobile_Device_Selection*.

Refer to the following table to know the variables and sample values:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the BitBar Cloud.	BitBar:Google Pixel 2
bitbar.apikey	Specify the user token generated for your BitBar account to authenticate your connection with the BitBar Cloud.	LkBldnjcnzrIcwWZpCZZxy

Variable	Action	Sample value
bitbar.host	Specify the host name of the BitBar Cloud instance.	appium.bitbar.com
bitbar.project	Specify the name of the project that contains the recorded test.	PlaybackMobile
bitbar.testrun	Specify a name for the test run that must be displayed in the BitBar dashboard for the test run.	CLIExecution
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for BrowserStack Cloud

You can create an XML file as a variable file that specifies the details of the BrowserStack Cloud.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices and applications.

Restriction: Resource Monitoring metrics of mobile device resources are not collected when the tests are run as an AFT suite.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inits id="N2D6E3E008FC11EDAE07D38ACD9588FD" type="com.ibm.rational.test.ft.aftsuite">
<group>
<tests>
<test path="/UI1/Tests/apidemos1.testsuite"/>
</tests>
```

```
<devices>
<device id="BrowserStack : Google Pixel 7 Pro" />
<device id="BrowserStack : Samsung Galaxy Tab S8" />
</devices>
</group>
</inits>
```

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
device_id	Specify the name of the mobile device that is connected to the BrowserStack Cloud.	BrowserStack : Google Pixel 7 Pro
browser-stack.project	Specify the name of the project that contains the recorded test.	PlaybackMobile
testpath	Specify a path for the test run that must be displayed in the BrowserStack dashboard for the test run.	CLIExecution
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for Perfecto Cloud

You can create an XML file as a variable file that specifies the details of the Perfecto Cloud.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

Restriction: Resource Monitoring metrics of mobile device resources are not collected for mobile tests in an AFT suite.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<Perfecto device name>" name="Mobile_Device_Selection"/>
<variable_init value="<security token>" name="perfecto.securitytoken"/>
<variable_init value="<perfecto host URL>" name="perfecto.host"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to Perfecto Cloud, create the XML file without the variable *Mobile_Device_Selection*.

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the Perfecto Cloud.	Perfecto:R48904TNSAZ
perfecto.securitytoken	Specify the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.	LkBldnjcnzrIcwWZpCZZxy
perfecto.host	Specify the URL of the Perfecto Cloud.	partners.perfectomobile.com
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for pCloudy Cloud

You can create an XML file as a variable file that specifies the details of the pCloudy Cloud.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

Restriction: Resource Monitoring metrics of mobile device resources are not collected for mobile tests in an AFT suite.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<pCloudy device name>" name="Mobile_Device_Selection"/>
<variable_init value="<api key>" name="pcloudy.apikey"/>
<variable_init value="<pcloudy host URL>" name="pcloudy.host"/>
<variable_init value="<pcloudy user name>" name="pcloudy.username"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to PCloudy Cloud, create the XML file without the variable *Mobile_Device_Selection*.

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the pCloudy Cloud.	pCloudy:SAMSUNG_GalaxyS8_Android_7.0.0_-3ced4
pcloudy.apikey	Specify the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.	csv7wxjyyzzgrzbqym62cvfg
pcloudy.host	Specify the URL of the pCloudy Cloud instance.	device.pcloudy.com
pcloudy.username	Specify the user name of your pCloudy account.	myUserName@pcloudy
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is support-	true

Variable	Action	Sample value
	ed only when you run a single test on a single device.	
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file to run mobile tests on multiple mobile clouds

When you want to run mobile tests on devices that are connected to multiple mobile clouds, you can create an XML file and use it as a variable file. You must specify the details of each of the mobile clouds in the variable file that enable Rational® Functional TesterHCL OneTest™ UI to connect to the mobile clouds.

You can configure details of the mobile clouds as an xml file. You can use the following sample XML file to enter the details of the name and value pairs for the mobile clouds.

For example, if you want to run on the BitBar Cloud and pCloudy Cloud, edit the XML file and remove the details of the Perfecto Cloud.

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BitBar API key>" name="bitbar.apikey"/>
<variable_init value="<BitBar host URL>" name="bitbar.host"/>
<variable_init value="<project name>" name="bitbar.project"/>
<variable_init value="<test name>" name="bitbar.testrun"/>
<variable_init value="<api key>" name="pcloudy.apikey"/>
<variable_init value="<pcloudy host URL>" name="pcloudy.host"/>
<variable_init value="<pcloudy user name>" name="pcloudy.username"/>
<variable_init value="<security token>" name="perfecto.securitytoken"/>
<variable_init value="<perfecto host URL>" name="perfecto.host"/>
</inits>
```

Note: You must only enter the details of the mobile clouds that you want to use.

Refer to the following table to know the details of the mobile clouds, that you need to enter in the XML file:

Mobile clouds	Name of the variable	Description
BitBar Cloud	*bitbar.apikey*	Specifies the user token generated for your BitBar account to authenticate the connection with the BitBar Cloud.

Mobile clouds	Name of the variable	Description
	bitbar.host	Specifies the host name of the BitBar Cloud instance.
	bitbar.project	Specifies the name of the project that contains the recorded test.
	bitbar.testrun	Specifies a name for the test run that must be displayed in the BitBar dashboard for the test run.
pCloudy Cloud	*pcloudy.apikey*	Specifies the API key of your pCloudy account to authenticate the user name for a successful connection.
	pcloudy.host	Specifies the URL of the pCloudy Cloud instance.
	pcloudy.username	Specifies the user name of your pCloudy account.
Perfecto Cloud	*perfecto.securitytoken*	Specifies the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.
	perfecto.host	Specifies the URL of the Perfecto Cloud.

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Related information

Configuration of test runs from the command line on page 1381

Running mobile tests for iOS applications from the command-line

You can run a mobile test without using the desktop client by using the command-line interface. You must use the **-varfile** parameter that specifies the complete path to the XML file in the command. The XML file contains the name-value pairs of the variables. The variables specify the path to the computer to which the iOS device or simulator is connected, and the other configurations required to run the mobile test on the connected device.

Running the command

Before you can run the command to run mobile tests from the command line, you must create a variable file that contains the details of the computer to which the mobile device is connected. You must have also recorded the mobile test.

You can issue the following command to run a mobile test from the command-line:

cmdline -workspace <workspace_full_path> -project <proj_relative_path> -eclipsehome <eclipse_full_path> -plugins <plugin_full_path> -suite <suite_relative_path> -importzip <full_path.zip> -varfile <variable_file_full_path> -configfile <file_full_path> -results <result_file> -overwrite <{"true" | "false"}> -quiet -vmargs <JVM_args> -publish <serverURL#project.name=projectName -publish_for {ALL,PASS,FAIL,ERROR, INCONCLUSIVE}> -labels <labelname1, labelname2> -exportlog <log_full_path> -exportstats <local_dir_path> -exportstatshtml <local_dir_path>

-exportstatsformat <name of the file format> -compare <"result_path1, result_path2"> -exportstatreportlist <stats_list> -execsummary <local_dir_path> -execsummaryreport <reportID> -usercomments <"any user comment"> -publishreports <"FT, STATS, TESTLOG"> -stdout -swapdatsets <existing_dataset_file_path:new_dataset_file-path>

Important: When you use the command to run a mobile test, you must ensure that the following conditions are satisfied:

- Use the variable file that contains the details of the computer, server, or cloud to which the iOS devices or simulators are connected.
- Specify the details of the variable file in the command.

Creating a variable file

You must create a variable file that contains the variable names and values as pairs that are required to connect to the iOS device or simulator. You can connect and run the mobile tests on iOS devices or simulators that are connected to the following computers, servers, or mobile clouds:

- Computer on which you have installed Rational® Functional TesterHCL OneTest™ UI on page 1425
- UI Test Agent on page 1426
- BitBar Cloud on page 1427
- BrowserStack Cloud on page 1428
- Perfecto Cloud on page 1429
- pCloudy Cloud on page 1430
- Variable file to run mobile tests on multiple mobile clouds on page 1431

Variable file for a computer on which you have installed Rational® Functional TesterHCL OneTest™ UI

You can create a variable file in XML format to specify the details required to run the mobile test on selected device.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<iOS device name>" name="Mobile_Device_Selection"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the iOS device or simulator that is connected to the computer.	Simulator:iPhone 11 Pro_14.0
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for UI Test Agent

You can create an XML file as a variable file that specifies the details of the remote computer on which you have installed the UI Test Agent and connected the iOS device or a simulator.

You can also add the details to enable the Rational® Functional TesterHCL OneTest™ UI to collect metrics for certain parameters of the resources used by the mobile devices.

A sample of the syntax or structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<iOS device or simulator name>" name="Mobile_Device_Selection"/>
<variable_init value="<UI Test Agent host URL>" name="appium.server.host"/>
<variable_init value="<port number>" name="appium.server.port"/>
<variable_init value="true" name="rm.selected"/>
<variable_init value ="any value equal to or above 1000" name="rm.polling.interval"/>
</inits>
```

Restriction: Resource Monitoring metrics of iOS simulators are not collected nor graphically displayed.

Refer to the following table to know the variables and values for the attributes mentioned in the XML sample:

Variable	Action	Sample value
device_id	Specify the name of the iOS device or a simulator that is connected to the computer.	Simulator:iPhone 11 Pro_14.0
appium.server-.host	Specify the host name or IP address of the remote computer on which you have installed the UI Test Agent. **Note:** The default value for this variable is 127.0.0.1. If no value is specified, the default value is used during the playback.	10.115.50.61
appium.server-.port	Specify the port number of the UI Test Agent that is installed on the remote machine.	7082
rm.selected	Optional. Set the value to *true* if you want to collect the metrics for certain parameters of resources used by mobile applications or devices. The collection of Resource Monitoring metrics and displaying of the data graphically is supported only when you run a single test on a single device.	true
rm.polling.interval	Specify the interval in milliseconds to collect the metrics for certain parameters of resources used by mobile applications or devices. The minimum value that you can enter is 1000 milliseconds. Enter the value, if you set the attribute *rm.selected* to *true*.	1000

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for BitBar Cloud

You can create an XML file as a variable file that specifies the details of the BitBar Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BitBar device name>" name="Mobile_Device_Selection"/>
<variable_init value="<BitBar API key>" name="bitbar.apikey"/>
<variable_init value="<BitBar host URL>" name="bitbar.host"/>
<variable_init value="<project name>" name="bitbar.project"/>
```

```
<variable_init value="<test name>" name="bitbar.testrun"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to BitBar Cloud, create the XML file without the variable *Mobile_Device_Selection*.

The following table lists the variables and the sample values that are used to create a variable file:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the BitBar Cloud.	BitBar:iPhone 11 Pro_14.0
bitbar.apikey	Specify the user token generated for your BitBar account to authenticate your connection with the BitBar Cloud.	LkBldnjcnzrIcwWZpCZZxy
bitbar.host	Specify the host name of the BitBar Cloud instance.	appium.bitbar.com
bitbar.project	Specify the name of the project that contains the recorded test.	PlaybackMobile
bitbar.testrun	Specify a name for the test run that must be displayed in the BitBar dashboard for the test run.	CLIExecution

The example variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="BitBar:iPhone 11 Pro_14.0" name="Mobile_Device_Selection"/>
<variable_init value="LkBldnjcnzrIcwWZpCZZxy" name="bitbar.apikey"/>
<variable_init value="appium.bitbar.com" name="bitbar.host"/>
<variable_init value="PlaybackMobile" name="bitbar.project"/>
<variable_init value="CLIExecution" name="bitbar.testrun"/>
</inits>
```

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for BrowserStack Cloud

You can create an XML file as a variable file that specifies the details of the BrowserStack Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BrowserStack user name>" name="browserstack.username"/>
<variable_init value="<BrowserStack host URL>" name="browserstack.host"/>
```

```
<variable_init value="<BrowserStack API key>" name="browserstack.apikey"/>
<variable_init value="<project name>" name="browserstack.project"/>
<variable_init value="<Browser(BrowserStack : device name)>" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

The following table lists the variables and the sample values that are used to create a variable file:

Variable	Action	Sample value
browserstack.username	Specify the user name of your BrowserStack account to authenticate your connection with the BrowserStack Cloud.	tester_1
browserstack.apikey	Specify the user token generated for your BrowserStack account to authenticate your connection with the BrowserStack Cloud.	api-cloud.browserstack.com
browserstack.host	Specify the host name of the BrowserStack Cloud instance.	KBk7PjhpqgfeVmxasUye
browserstack.project	Specify the name of the project that contains the recorded test.	CMDLINETest
RTW_WebUI_Browser_Selection	Specify the browser to be used on the mobile device that is connected to the BrowserStack Cloud.	Chrome(BrowserStack : iPhone 11)

The example variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="tester_1" name="browserstack.username"/>
<variable_init value="api-cloud.browserstack.com" name="browserstack.host"/>
<variable_init value="KBk7PjhpqgfeVmxasUye" name="browserstack.apikey"/>
<variable_init value="CMDLINETest" name="browserstack.project"/>
<variable_init value="Chrome(BrowserStack : iPhone 11)" name="RTW_WebUI_Browser_Selection"/>
</inits>
```

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for Perfecto Cloud

You can create an XML file as a variable file that specifies the details of the Perfecto Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<device name>" name="Mobile_Device_Selection"/>
<variable_init value="<security token>" name="perfecto.securitytoken"/>
```

```
<variable_init value="<perfecto host URL>" name="perfecto.host"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to Perfecto Cloud, create the XML file without the variable *Mobile_Device_Selection*.

The following table lists the variables and the sample values that are used to create a variable file:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the Perfecto Cloud.	Perfecto:8D3E35CF16D8D827E4827ABBCD0E582E2761CADA
perfecto.securitytoken	Specify the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.	LkBldnjcnzrIcwWZpCZZxy
perfecto.host	Specify the URL of the Perfecto Cloud that is configured to communicate with Rational® Functional TesterHCL OneTest™ UI.	partners.perfectomobile.com

The example variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="Perfecto:8D3E35CF16D8D827E4827ABBCD0E582E2761CADA"
 name="Mobile_Device_Selection"/>
<variable_init value="LkBldnjcnzrIcwWZpCZZxy" name="perfecto.securitytoken"/>
<variable_init value="partners.perfectomobile.com" name="perfecto.host"/>
</inits>
```

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file for pCloudy Cloud

You can create an XML file as a variable file that specifies the details of the pCloudy Cloud. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<device name>" name="Mobile_Device_Selection"/>
<variable_init value="<api key>" name="pcloudy.apikey"/>
<variable_init value="<pcloudy host URL>" name="pcloudy.host"/>
<variable_init value="<pcloudy user name>" name="pcloudy.username"/>
</inits>
```

Note: When you want to run an AFT Suite on devices connected to pCloudy Cloud, create the XML file without the variable *Mobile_Device_Selection*.

The following table lists the variables and the sample values that are used to create a variable file:

Variable	Action	Sample value
Mobile_Device_Selection	Specify the name of the mobile device that is connected to the pCloudy Cloud.	pCloudy:Apple_iPhone8plus_Ios_14.4.0_9db66
pcloudy.apikey	Specify the API key of your pCloudy account to authenticate the user name for a successful connection.	csv7wxjyyzzgrzbqym62cvfg
pcloudy.host	Specify the URL of the pCloudy Cloud instance.	device.pcloudy.com
pcloudy.username	Specify the user name of your pCloudy account.	myUserName@pcloudy

The example variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="pCloudy:Apple_iPhone8plus_Ios_14.4.0_9db66" name="Mobile_Device_Selection"/>
<variable_init value="csv7wxjyyzzgrzbqym62cvfg" name="pcloudy.apikey"/>
<variable_init value="device.pcloudy.com" name="pcloudy.host"/>
<variable_init value="myUserName@pcloudy" name="pcloudy.username"/>
</inits>
```

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Variable file to run mobile tests on multiple mobile clouds

When you want to run mobile tests on devices that are connected to multiple mobile clouds, you can create an XML file and use it as a variable file. You must specify the details of each of the mobile clouds in the variable file that enable Rational® Functional TesterHCL OneTest™ UI to connect to the mobile clouds.

You can configure details of the mobile clouds as an xml file. You can use the following sample XML file to enter the details of the name and value pairs for the mobile clouds.

For example, if you want to run on the BitBar Cloud and pCloudy Cloud, edit the XML file and remove the details of the Perfecto Cloud.

```
<?xml version="1.0" encoding="UTF-8"?>
<inits>
<variable_init value="<BitBar API key>" name="bitbar.apikey"/>
<variable_init value="<BitBar host URL>" name="bitbar.host"/>
<variable_init value="<project name>" name="bitbar.project"/>
<variable_init value="<test name>" name="bitbar.testrun"/>
<variable_init value="<api key>" name="pcloudy.apikey"/>
```

```
<variable_init value="<pcloudy host URL>" name="pcloudy.host"/>
<variable_init value="<pcloudy user name>" name="pcloudy.username"/>
<variable_init value="<security token>" name="perfecto.securitytoken"/>
<variable_init value="<perfecto host URL>" name="perfecto.host"/>
</inits>
```

Note: You must only enter the details of the mobile clouds that you want to use.

Refer to the following table to know the details of the mobile clouds, that you need to enter in the XML file:

Mobile clouds	Name of the variable	Description
BitBar Cloud	*bitbar.apikey*	Specifies the user token generated for your BitBar account to authenticate the connection with the BitBar Cloud.
	bitbar.host	Specifies the host name of the BitBar Cloud instance.
	bitbar.project	Specifies the name of the project that contains the recorded test.
	bitbar.testrun	Specifies a name for the test run that must be displayed in the BitBar dashboard for the test run.
pCloudy Cloud	*pcloudy.apikey*	Specifies the API key of your pCloudy account to authenticate the user name for a successful connection.
	pcloudy.host	Specifies the URL of the pCloudy cloud instance.
	pcloudy.username	Specifies the user name of your pCloudy account.
Perfecto Cloud	*perfecto.securitytoken*	Specifies the user token generated for your Perfecto account to authenticate your connection with the Perfecto Cloud.
	perfecto.host	Specifies the URL of the Perfecto Cloud.

You can then use the variable file with the **-varfile** option in the command to run a test from the command line.

Related information

Configuration of test runs from the command line on page 1381

Running Windows tests from the command-line interface

You can run a Windows test without using the desktop client by using the command-line interface. You can either run the Windows test on the local computer or on a remote computer.

Running the command

If you want to run the Windows test on your local computer, then you can run the following command from the command-line:

Note: You must have recorded the Windows test.

```
cmdline -workspace workspace_full_path -project proj_rel_path -suite suite_rel_path
```

Creating a variable file

You need to create a variable file only when you want to run the Windows test in the following scenarios:

- On the local computer that has a different port number on page 1433
- On the remote computer that has a different port number and host number on page 1434

The **-varfile** parameter specifies the complete path to the XML file in the command. The XML file contains the name-value pairs of the variables.

You must include the **-varfile** parameter in the following command when you run from the command-line:

```
cmdline -workspace workspace_full_path -project proj_rel_path -suite suite_rel_path -varfile
variable_file_full_path
```

Note: The variable file is not required when you run Windows test on a local computer that contains the same port number.

Variable file for the local computer on which you want to run Windows test

Create an XML file as a variable file that specifies the details of the local computer on which you have installed the Windows application and the UI Test Agent. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<variable_init name="appium.server.port" value=<"port number"/>
</inits> </inits>
```

The following table lists the variable and the action required for the value field:

Name of the variable	Description
appium.server.port	Specifies the port number of the UI Test Agent that is installed on your local machine.

For example,

```
<?xml version="1.0" encoding="UTF-8"?>
<variable_init name="appium.server.port" value="7082"/>
</inits>
```

Variable file for the remote computer on which you want to run Windows test

Create an XML file as a variable file that specifies the details of the remote computer on which you installed the Windows application and the UI Test Agent. The format of the variable file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<inits xmlns=<"Windows application path">
<variable_init name="appium.server.host” value=<"host number"/>
<variable_init name="appium.server.port" value=<"port number"/>
</inits>
```

The following table lists the variable and the action required for the value field:

Name of the Variable	Description
appium.server.host	Specifies the IP address of the remote computer on which the UI Test Agent and the Windows applications are installed.
appium.server.port	Specifies the port number of the UI Test Agent that is installed on the remote machine.

For example,

```
<?xml version="1.0" encoding="UTF-8"?>
<inits xmlns="http://www.ibm.com/rational/test/lt/varinit">
<variable_init name="appium.server.host" value="10.115.160.148"/>
<variable_init name="appium.server.port" value="7082"/>
</inits>
```

Related information

Configuration of test runs from the command line on page 1381

Configuration of test runs from the Functional Test perspective

When you created functional tests, you must configure the run of the recorded functional tests from the Functional Test perspective before you can view the test results.

You can run functional test scripts as part of your automated testing. See Script Playback on page 1436. You can run scripts as part of any of the following test phases:

- Test development phase on page 1437
- Regression testing phase on page 1437

Prerequisites for functional test runs

You must have recorded test scripts in the Functional Test perspective of IBM® Rational® Functional TesterHCL OneTest™ UI. See Testing with Functional Test perspective.

Before you can configure a test run of a functional test, you must ensure that the application-under-test is in the same state that it was in when you recorded the script. See Restoring the test environment before playback on page 1437.

You can configure how unexpected windows are to be handled so that scripts can be played back smoothly without interruption. See Configuring how to handle unexpected windows during playback on page 1438.

If changes are made to the application-under-test after it was recorded, there might changes in the object hierarchy when you run such a test. You can avoid test failures by performing any of the following tasks:

- Insert dynamic objects. See Inserting dynamic test objects on page 1439.
- Enable the dynamic find feature when you configure a test run. See Enabling the dynamic find feature on page 1440.
- Use ScriptAssure. See Using ScriptAssure on page 1441.

If there are multiple instance of the applications under test, which might cause Rational® Functional TesterHCL OneTest™ UI to not identify the correct application, you can prevent or enable Rational® Functional TesterHCL OneTest™ UI to deal with this ambiguous recognition situation. See Ambiguous object recognition in functional testing on page 1443.

Monitoring functional test runs

You can find information about monitoring test runs of a functional test:

- You can use the playback monitor to view the test details as the test is played back. See Playback Monitor on page 1443.
- When you want to pause or stop the play back at any time after you started a run of the recorded script, you can pause or stop the test run. See Pausing or stopping script playback on page 1444.

Running functional tests

You can find information about the following tasks that you can configure and run from the Functional Test perspective:

- Running scripts from Rational Functional TesterHCL OneTest UI on page 1445
- Playback of test scripts for HTML applications by using the Web UI engine on page 1447
- Running a script from the Microsoft Edge browser on page 1449

Analyzing test runs

After you run tests from the Functional Test perspective, you can perform the following tasks to debug or view the steps that failed in the test run:

- Debugging scripts on page 1449
- Screen snapshot on playback failure of functional tests on page 1450

Script Playback

When you play back a script, Rational® Functional TesterHCL OneTest™ UI replays your recorded actions, which automates the software testing cycle.

Such automation allows you to test each new build of your application faster and more thoroughly than by manual testing, reducing testing time and increasing both coverage and overall consistency.

There are two general phases of script playback:

In the Test Development Phase on page 1437 you play back scripts to verify that they work as intended, using the same version of the application-under-test that you used to record. This phase validates the expected behavior for the application.

In the Regression Testing Phase on page 1437 you play back scripts to compare the latest build of the application to the baseline established during the test development phase. Regression testing identifies differences that may have been introduced since the last build. You can evaluate these differences to determine whether they are defects or changes.

Note:

You can run a script for these purposes:

- To play back a script locally from Rational® Functional TesterHCL OneTest™ UI or from Engineering Test Management.
- To play back a script remotely on different computers and platforms from Engineering Test Management.
- To play back a script to verify that it works on different Java™ environments or browsers.
- To debug a script.

Note: The license validation also happens during the functional test script playback. If you are using floating or token licenses, ensure that the required licenses to work with Rational® Functional TesterHCL OneTest™ UI is available.

You can use Rational® Functional TesterHCL OneTest™ UI plug-in for UrbanCode™ Deploy to continuously initiate the launch of functional tests. For more information see: UrbanCode: Running functional tests. Also, while running test scripts through command-line or through UrbanCode™ Deploy, you must ensure that at any given time, only a single playback process runs on the machine hosting Rational® Functional TesterHCL OneTest™ UI.

If the recording was performed in one document mode on Internet Explorer, the play back needs to be done on the same document mode in Internet Explorer. Using different document modes may render the controls differently and playback may vary.

Test development phase

In the Test Development Phase, you play back scripts to verify that they work as intended, using the same version of the application-under-test that you used to record. This phase validates the expected behavior of the application.

The test development phase consists of six steps:

1. Restore the test environment and set the playback options.
2. Play back each script against the same version of the application-under-test that was used for recording to verify that the script performs as intended.
3. Analyze the results in the Rational® Functional TesterHCL OneTest™ UI log.
4. Use the Rational® Functional TesterHCL OneTest™ UI Verification Point Comparator to determine the cause of verification point failures.
5. If the script fails, edit, debug, or re-record it so that it runs as required.

Regression testing phase

When you have a baseline of expected behavior for your application-under-test, you can automate regression testing for subsequent builds.

The regression testing phase consists of five steps:

1. Restore the testing environment and set the playback options.
2. Play back the test script against a new build of the application-under-test.
3. Analyze the results in the Rational® Functional TesterHCL OneTest™ UI log.
4. Use the Rational® Functional TesterHCL OneTest™ UI Verification Point Comparator to determine the cause of verification point failures. If verification points fail because of intentional changes to the application-under-test, update the baseline data using the Comparator.
5. If necessary, revise your test scripts to use new features in the application-under-test. Then play back the revised test scripts against the current build and re-evaluate the results.

Restoring the test environment before playback

The state of both the environment and the application-under-test can affect script playback. If the recorded environment and the playback environment are different, playback problems can occur.

About this task

Before playing back a script, verify that your application-under-test is in the same state that it was in when you recorded the script. Any applications and windows that were open, displayed, or active when you started recording the script must be open, displayed, or active when you start playback. In addition, restore any relevant network settings, active databases, and system memory to the same state as when the script was recorded.

Also, ensure that the scale and layout in the display settings of the system match with the settings used to create the test. The zoom level must always be 100%.

Note: Multiple open instances of the application-under-test will cause an ambiguous recognition error during playback. For more information about ambiguous recognition, see Ambiguous Recognition window on page 1443.

As part of your test process, you might want to do cross-JVM testing to verify that your applications run correctly under various JVMs. For information about supported JVMs, see Java Support on page 1716.

1. Enable your web browsers and Java™ environments.
2. Use the Configuration Editor to configure your applications for recording and playback.
3. Set the appropriate playback preferences.

Note: The license validation also happens during the functional test script playback. If you are using floating or token licenses, ensure that the required licenses to work with Rational® Functional TesterHCL OneTest™ UI are available.

Configuring how to handle unexpected windows during playback

Problems can occur during script playback if unexpected windows are displayed. Unexpected windows such as security and warning message dialog boxes or custom dialog boxes can cause playback failure with exceptions such as `object not found` or `window not activated`. You can configure how unexpected windows can be handled so that scripts can be played back smoothly without interruption.

About this task

You can identify unexpected windows in each test domain and specify how such windows must be handled during script playback.

1. Click **Configure > Configure Unexpected Windows** to open the Configure Handling of Unexpected Windows dialog box.
2. From the **Test Domain** list, select the domain for which you are configuring unexpected windows.

   If you do not know the domain of the unexpected window, select **All Windows** to see a master list of unexpected windows with window titles for all domains. The **Unexpected Window Title** field lists all unexpected windows in the selected domain.

   If the required window title is not listed in the **Unexpected Window Title** list, you can add the missing window to the list by clicking the **Add Window** option. An empty row is added in the Unexpected Window Title list. Double-click the cell in the empty row and type the window title. To remove a window from the list, select the corresponding window row and click **Remove Window**.
3. Select the window for which you want to configure the action to be taken. You can configure an unexpected window either to be closed automatically, or have a specific action performed on it, determined by additional recognition properties that you can define. Do one of these procedures:

a. To indicate that the window must be closed automatically, select Close from the **Select Action** list.

b. To indicate that a specific action must be performed on the window, select Click Action from the **Select Action** list to indicate that a specific action be performed on a specific control on the window.

   The **Configure action objects properties** field displays the object recognition properties of the control on the selected window such as Property Name and Property Value. To modify the recognition properties, double-click the **Property Name** cell in the required row and type the required property name.

   If a required property name or value is not displayed, you can add the property by clicking **Add Property**. An empty row is added in the **Configure action objects properties** field. Double-click the **Property Name** cell in the empty row, and type the required property name. You can also use the Test Object Inspector to get properties for the control on unexpected window. Open the unexpected window and the Test Object Inspector. Move the cursor over the window to get the title. Move the cursor over the specific control, to get its property name and value. See Displaying test object information on page 885 for instructions to use the Test Object Inspector.

   To remove recognition properties for the control on the selected window, select the required property row and click **Remove Properties**.

   To apply modified recognition properties to the control on the selected window, click **Apply**.

4. Select the **Perform close action for 'non-configured' windows** check box to set the Close action for all windows that have not been configured. You can use this setting to close any unexpected windows that were not captured during recording of the script.
5. After completing your configuration, click **Finish** to save your changes.

What to do next

Enable the unexpected window handling feature by selecting the **Enable handling of unexpected windows** check box on the Unexpected Windows page in the Preferences dialog box. When you play back a script, displayed unexpected windows are handled according to the configuration on the Configure Handling of Unexpected Windows dialog box.

Inserting dynamic test objects

You can also insert dynamic test objects by using **Insert Dynamic Test Object**. The hierarchy of a test object in the object map represents the order of search of the particular control during playback. Over a series of application changes, the hierarchy of the objects may change if new objects are introduced in the test application. This results in a playback failure. Using dynamic test objects you can anchor a test object as a descendant to its parent.

1. From the test object map menu, click **Applications > Run** to open the Select an Application dialog box.
2. In the **Application Name** field, select the application that contains the controls you want to test and click **OK**.
3. In the Test Object Map toolbar, click **Test Object > Insert Dynamic Test Object**  .
   Result

Rational® Functional TesterHCL OneTest™ UI opens the Object Map dialog box.

4. On the **Select an Object** page, click the **Object Finder** icon and drag it into the application over the object you want to add to the test object map. For other methods of selecting objects, see Select an Object on page 1823.
5. Click **Next**.
6. In the **Add Dynamic Test Object** dialog box, select **Anchor to Selected Parent**.
   By selecting **Anchor to Selected Parent**, you are making the new object a descendant of its parent. You can now search for the object dynamically, anchoring to the parent. You can edit the recognition properties by double clicking on the object properties.
7. Select the object that you want to insert and click **Finish**.

Note: To convert an existing mapped object to a dynamic object, right-click in the test object map and click **Convert To Dynamic Test Object**. The Administrative property displays an additional `descriptionobject` property. To convert a dynamic test object to a mapped test object, set the `descriptionobject` property to false. However, you must ensure that the test object is a mappable child of its parent.

Note: To prevent playback failure due to object hierarchy changes, you can also enable the dynamic find feature, which enables Rational® Functional TesterHCL OneTest™ UI to locate test objects in the application-under-test whose hierarchical position may have been altered from the position in the test object map. For information about the dynamic find feature, see Enabling the dynamic find feature on page 1440.

Enabling the dynamic find feature

The test object map lists in a hierarchy the test objects in the application under test. Changes to the application-under-test might result in changes in the object hierarchy. During playback, Rational® Functional TesterHCL OneTest™ UI is then unable to find an object whose hierarchical position has changed, and this causes playback failure. With the dynamic find feature you can prevent playback failure that results from hierarchy changes in the application under test. The dynamic find feature performs searches for objects whose hierarchy has changed, when a search that is based on object recognition scoring (ScriptAssure) fails to find such objects.

About this task

On the **Dynamic Find Enablement** page in the **Preferences** dialog, you can enable or disable the dynamic find feature for all functional test scripts in the integrated development environment (IDE). For an individual script, you can enable or disable the feature on the **Select Log** page. For instructions, see Select Log page on page 1831. You can also enable or disable the feature from the command-line interface. For information, see Command line usage examples on page 1730.

Note: When you enable the dynamic find feature in the **Preferences** dialog, the setting applies to all the scripts in the IDE. You can override this preference for an individual script on the **Select Log** page, when you run the script.

1. In the product menu, click **Window > Preferences** to open the Preferences dialog box.
2. Expand **Functional Test**, and then click **Playback**.
3. Click **Dynamic find enabled**.
4. Select the **Enable script find if scoring find fails** check box.

Note: To prevent playback failure because of hierarchy changes, you can also use the Insert Dynamic Test Object method. Using this method, you can anchor a test object as a descendant of its parent. This renders script playback resilient to object hierarchy changes.

Related information

Inserting dynamic test objects on page 1439

Enabling the dynamic find feature in the Visual Studio IDE

The Rational® Functional TesterHCL OneTest™ UI test object map lists the test objects in the application under test, in a hierarchy. Changes to the application under test might result in changes in object hierarchy. During playback, Rational® Functional TesterHCL OneTest™ UI is unable to find an object whose hierarchical position has changed, and this causes playback failure. With the dynamic find feature, you can prevent playback failure that results from hierarchy changes in the application under test. The dynamic find feature performs searches for objects whose hierarchy has changed when a search based on object recognition scoring (ScriptAssure) fails to find such objects.

About this task

On the Select Log page, you can enable or disable the dynamic find feature for an individual script when you run the script.

1. Run the script for which you want to enable dynamic find.
2. On the Select Log page, select the **Enable script find if scoring find fails** check box.

Using ScriptAssure

Using ScriptAssure™, you can play back scripts successfully even when the application-under-test has been updated.

Each object in a test object map has a set of recognition properties, which are typically established during recording. For example, a button has five recognition properties: name, type, role, class, and index. To find an object in the application-under-test during playback, Rational® Functional TesterHCL OneTest™ UI compares the object in the application with recognition properties in the test object map.

Each property of a test object has an associated recognition weight value, which is a number from 0 to 100. Rational® Functional TesterHCL OneTest™ UI uses the weight value for each recognition property to determine the importance of the property. For example, the name, type, role, and class recognition properties of the button object have a weight of 100; the class recognition property has a weight of 50.

Rational® Functional TesterHCL OneTest™ UI uses criteria to assign a recognition score to objects in the application-under-test. For example, if the object exactly matches the recognition properties in the test object map, its score is 0. If the object has one property with a weight of 100 that does not match, its score is 10,000. If the object has two properties that do not match, its score is 20,000, and so on. The higher the recognition score, the less exact the match.

For Rational® Functional TesterHCL OneTest™ UI to recognize an object in the application-under-test, the object properties must match the properties recorded in the test object map. If the object properties do not match and the weight of the recognition property is less provided that their score lies within 10,000, Rational® Functional TesterHCL OneTest™ UI still proceeds with the test. If the score exceeds the value of 10,000 but less than the default threshold of 20,000, Rational® Functional TesterHCL OneTest™ UI writes a weak recognition warning to the log.

You can also enter values to set thresholds for recognition scores, such as the maximum acceptable recognition score, last chance recognition score, ambiguous recognition scores difference threshold, and warn if accepted score is greater than. During playback, the recognition scores for a test object's recognition properties are added and the total compared to the thresholds set in the **ScriptAssure** Page.

If objects in the application-under-test have changed, you can still play back scripts in Rational® Functional TesterHCL OneTest™ UI by using the ScriptAssure™ feature to control object-matching sensitivity.

You can use ScriptAssure™ in two ways:

Standard -- The ScriptAssure Page-Standard on page 841 controls object-matching sensitivity during playback by using a slider control. To set the tolerance for differences between the object in the application-under-test, you move the **Recognition Level** slider between **Strict** and **Tolerant**. To find differences between the object and the recognition properties, you move the **Warning Level** slider between **High** and **None**.

Advanced -- The ScriptAssure Page-Advanced on page 840 sets thresholds for recognition scores. You can set a maximum score to consider a test object as a candidate for recognition; you can also request warnings when candidate objects have a score higher than the designated threshold.

Tips for using ScriptAssure

- If you want the script to play back faster and with fewer warnings, set the thresholds high. The recognition is less fussy but more prone to error. This behavior might be useful in some situations.
- If recognition is weak, examine your test object map. Have accessible names changed? (For example, is "Place Order" now "Place Your Order?") If the application has changed permanently, update the test object map to reflect the change. In an internationalization situation, change the label of the test object, not its accessible name.

- If the application has a dynamic object or if several versions of the application are slightly different, correct versions of an object, replace the recognition property with a regular expression. You can also use a numeric range to accept more than one value of a property. For information, see Replacing an Exact-Match Property with a Pattern.
- If it is late in the development cycle and you are doing maintenance, verify that your scripts work and have the best possible recognition by setting the warning level to **High**. You will receive warnings about possible problem areas, and if you do, fix the map.

Ambiguous object recognition in functional testing

Ambiguous recognition occurs when Rational® Functional TesterHCL OneTest™ UI can not uniquely identify an object in the system-under-test. This commonly happens when Rational® Functional TesterHCL OneTest™ UI cannot differentiate between an instance of the application-under-test started by a script playback and an instance of the same application inadvertently left open previous to script playback. This also applies to identical windows from one application and identical HTML documents. Ambiguous recognition will cause script playback failure unless the duplicate application is closed.

If Rational® Functional TesterHCL OneTest™ UI finds more than one instance of the application-under-test during the playback of a script the **Ambiguous Recognition** window will open allowing you to close the duplicate instance and resume playback.

Preventing ambiguous recognition

One common cause of ambiguous recognition is residual windows left open from a previous playback of a test script.

To avoid this issue take the following actions:

- Make closing the application-under-test the last action recorded in the test script.
- If script playback fails, close all windows opened by script playback before replaying the script.

Dealing with ambiguous recognition

If the **Ambiguous Recognition** window opens correct the situation and restart playback.

The **Ambiguous Recognition** window opens and playback pauses.

1. Minimize open windows until the **Ambiguous Recognition** window is visible.
2. Find and close the duplicate application instance using the information in the **Ambiguous Recognition** window.
3. Click **OK** in the **Ambiguous Recognition** window to resume playback.

Playback Monitor

During playback you can view the script name, the number of the line that is executing, status icons, and a description of the action in progress from the Playback Monitor.

The Playback Monitor consists of four parts:

- Script name
- Description of action in progress
- Status icons
- Script line number
- Stop and pause or resume buttons

Playback Monitor icons

The Playback Monitor includes four status icons:

Status Icon	Description
	Rational® Functional TesterHCL OneTest™ UI is looking for an object in an application.
	Rational® Functional TesterHCL OneTest™ UI is not looking for an object in an application.
	Rational® Functional TesterHCL OneTest™ UI is waiting for an object in an application to appear or for a script delay to end.
	Rational® Functional TesterHCL OneTest™ UI is executing the current script.

Turning off the Playback Monitor

You can turn off the Playback Monitor.

By default, the Playback Monitor is on.

1. Click **Window > Preferences Tools > Options**.
2. In the left pane expand **Functional Test**, expand **Playback**, and click **Monitor**.
3. Clear the **Show monitor during playback** check box.

Pausing or stopping script playback

Playback of a script can be paused or stopped.

Pausing and resuming playback

About this task

The script is currently playing and the playback monitor is open.

1. Press F12 to pause playback.
2. Click the **resume** button on the playback monitor to restart playback.

Stopping script playback

About this task

The script is currently playing back and the Playback Monitor is open.

Press F11 to stop playback.

Note: If playback is paused you can click the **stop** button on the Playback Monitor to stop playback.

Results

Script playback ends and a failure is recorded in the playback log.

What to do next

Close the application that was associated with the script being played back.

Running scripts from Rational® Functional TesterHCL OneTest™ UI

You can run scripts from Rational® Functional TesterHCL OneTest™ UI so that all of your recorded actions are played back. The recorded actions include the actions that you performed to start and stop an application, add verification points, and any other actions that you performed while using the application.

Before you begin

You must have completed the following tasks:

- Read and been familiar with using the unexpected windows handling feature during the playback. See Configuring how to handle unexpected windows during playback on page 1438.
- Read and been familiar with enabling the dynamic find feature during the playback. See Enabling the dynamic find feature on page 1440.

1. Configure your application under test by setting the appropriate Java™ environment or web browser to run the application.
2. Run the script in any of the following ways:
    - Open the script from the **Functional Test Projects** view, and then click **Run Functional Test Script** in the Rational® Functional TesterHCL OneTest™ UI toolbar.
    - Right-click the script in the **Functional Test Projects** view, and then click **Run**.
    - Click the script in the **Functional Test Projects** view, and then click **Script > Run**.
    - Click the script in the Solution Explorer, and then click **Run Functional Test Script** in the Functional Test toolbar.
    - Right-click the script in the Solution Explorer, and then click **Run**.
    - Click the script in the Solution Explorer, and then click **Script > Run**.

    The **Select Log** dialog is displayed.
3. Retain the default log name or enter a log name for your script.
4. **Optional:** Select the following checkboxes, if they are not selected:

a. **Don't show this wizard again**: Select the checkbox to prevent the **Script Launch** wizard from starting each time you run the test script.
b. **Enable handling of unexpected windows**: Select the checkbox to enable the unexpected window handling feature for the test script.
c. **Enable script find if scoring find fails**: Select the checkbox to enable the dynamic find feature for the test script.

Note: You can enable any of these features in the **Windows > Preferences** page for all the scripts. The corresponding checkbox is automatically selected in the **Select Log** dialog for all the subsequent functional tests. You can clear a checkbox in the **Select Log** dialog, if you do not want to enable the feature for the script that you are running.

5. Click **Next**.
   Result
   The **Specify Playback Options** dialog is displayed.
6. **Optional:** Perform the following tasks in the **Specify Playback Options** dialog:
   a. Enter or select the command-line arguments in the **Run arguments** field to pass to the script. The most recently used command-line arguments appear in the list.
   b. Select a number or select **Iterate Until Done** in the **Dataset Iteration Count** field to specify the number of times the test script can run during the playback.
   c. Select the **Perform playback in interactive mode** checkbox to dynamically resolve the common runtime situations.
   d. Select the **Enable Response Time Breakdown** checkbox to generate performance metrics for the application under test.
7. Click **Finish** to run the test script.

Results

Rational® Functional TesterHCL OneTest™ UI is minimized, and the playback monitor is started. While the script plays back all of your recorded actions, the corresponding messages are displayed in the playback monitor.

What to do next

After the test playback is completed, you can view the results of the playback in the test log that is displayed in a separate browser window.

Note: If the playback monitor does not start, or the test log does not open in Rational® Functional TesterHCL OneTest™ UI, you can click **Windows > Preferences > Functional Test > Playback**, and then check the settings on the **Monitor** page and **Logging** page, respectively.

Related reference

Select Log page on page 1831

Related information

Playing back low level mouse and keyboard actions on page 1204

Pausing or stopping script playback on page 1444

Playback of test scripts for HTML applications by using the Web UI engine

You can play back test scripts for HTML applications in web browsers by using the Web UI engine without enabling the Java environments and browsers.

After you install or reinstall Rational® Functional TesterHCL OneTest™ UI, the **Play back with Web UI Extension** and **Play back with Web UI Action** checkboxes in **Windows > Preferences > Functional Test > Playback** are selected as the default options. Therefore, when you play back a functional test script for an HTML application, Rational® Functional TesterHCL OneTest™ UI automatically selects the Web UI engine for the playback.

You can also set a specific browser profile in Rational® Functional TesterHCL OneTest™ UI before you play back the test script for the HTML application. If you set the profile and then run the test script, the browser displays the profile details before the application is loaded. However, if you do not set any specific browser profile, then the test script runs on the browser with a temporary profile that is not configured with any predefined settings. See Setting a specific browser profile for the playback of functional HTML tests on page 783.

Notes:

- When you play back a functional test script for an HTML application in the Mozilla Firefox, Microsoft Edge, or Google Chrome browser by using the Web UI engine, and if there is a browser and driver incompatibility, then Rational® Functional TesterHCL OneTest™ UI automatically downloads the appropriate driver to play back the test scripts successfully.

- You can clear the cache and history of the Mozilla Firefox, Microsoft Edge, and Google Chrome browsers for the functional test playback. You can go to **Window > Preferences > Test > Test execution > UI Test Playback > Desktop**, and then select the **Clear cache** and **Clear history** checkboxes before the test playback.

Related information

Enabling web browsers on page 780

Google Chrome browser support on page 785

Supported keyboard actions

You can use specific keyboard actions in the functional test scripts for HTML applications that are run by using the Web UI Engine.

You can add the inputKeys navigation action to the test script and provide the keyboard strokes to the navigation actions. Also, you can use the combination of special keys such as SHIFT or CTRL along with other keys.

The list of supported navigation key values and their formats are as follows:

Key value	Key format
PgUp	{extPgUp}
PgDn	{extPgDn}
Home	{extHome}
End	{extEnd}
Left arrow key	{extLeft}
Right arrow key	{extRight}
Up arrow key	{extUp}
Down arrow key	{extDown}
F1 to F12	{F1} ….{F12}

Restriction: If the functional test script has an input of the F12 key, then the default command mapped to the F12 key takes precedence over the action expected in the browser window when you run the test by using the Web UI Engine. You must reassign the key before you run such tests with function keys in the script.

For more information, refer to How to change the playback hot keys.

Running a script from the Microsoft™ Edge browser

You can use IBM® Rational® Functional TesterHCL OneTest™ UI to play back test scripts using the Microsoft™ Edge browser.

About this task

When you play back functional tests in the Edge browser by using the Web UI extension, if there is a browser and driver incompatibility, Rational® Functional TesterHCL OneTest™ UI automatically downloads the appropriate driver to play back tests successfully. For more information, see the related links.

1. Start the Edge browser using either `startBrowser()` or `startApp()`.
   Choose from:
   - To use `startBrowser()`: `startBrowser("Edge", "https://www.google.co.in/");`
   - To use `startApp()`:
     a. Click **Configure > Enable Environments for Testing > Web Browsers** and click **Add**.
     b. Browse to `C:\Program Files (x86)\Microsoft\Edge\Application` and click **Add**.
     c. Add an HTML application and select **Edge** as the browser in the **Configure > Configure Applications for Testing** wizard.
2. Add `sleep()` of at least 15 seconds after `startBrowser()` or `startApp()`.

Related information

Playback of test scripts for HTML applications by using the Web UI engine on page 1447

Debugging scripts

You can use the same process to debug a Functional Test script as you would to debug other Visual Basic or Java™ applications. If you prefer, in Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, you can debug your script in the Rational® Functional TesterHCL OneTest™ UI Debug Perspective which makes it easier to start the debugger that comes with the Java™ Development Toolkit.

About this task

In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, Functional Test scripts are recorded in the Java™ programming language. Debugging these scripts is the same process as debugging a conventional Java™ application. For more information about using the Test Debug Perspective, see the IBM® *Java™ Development User Guide* (Getting Started: Debugging Your Programs).

In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, Functional Test scripts are recorded in the Visual Basic programming language. Debugging these scripts is the same process as debugging a conventional Visual Basic application. For more information about debugging in Visual Basic, see the Microsoft® Developer Network (MSDN) *Developing with Visual Studio.NET* (Building, Debugging, and Testing). In Rational®

Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, more views appear when a script is in debugging mode.

Setting the Debug Perspective preference

1. Click **Window > Preferences**.
2. In the left pane, expand **Functional Test**.
3. Click **Workbench > Advanced**.
4. Clear the **Switch to Test Debug Perspective when debugging** check box.

Results

Debugging a script

1. Configure your application for testing by setting the appropriate Java™ environment or web browser as the default to run the application you are testing.
   For information, see *Before you record* related topic.
2. Open the Debug Perspective in any of the following ways:
   - In the Projects view, click a script and click the **Debug Functional Test Script** button  in the Functional Test toolbar. In the Solution Explorer, click a script and click the **Debug Functional Test Script** button  in the product toolbar.
   - In the Projects view, right-click a script and click **Debug**. In the Solution Explorer, right-click a script and click **Debug**.
   - In the Projects view, click a script and from the Functional Test menu, click **Script > Debug**. In the Solution Explorer, click a script and from the Rational® Functional TesterHCL OneTest™ UI menu, click **Script > Debug**.

   Result

   The Test Debug Perspective opens and provides information as the script plays back under the debugger.

   After the script runs, a log file is displayed. If a log is not displayed, check the settings in the Logging page.

   If a verification point fails, the Comparator appears so that you can analyze the results. If the Comparator does not start, check the settings in the Logging page.

Screen snapshot on playback failure of functional tests

If playback of a script causes an exception to be thrown, Rational® Functional TesterHCL OneTest™ UI takes a screen snapshot at the time of the failure. The screen snapshot is accessible through the log.

Accessing the screen snapshot in an HTML log type

HTML is the preferred log type to access the screen snapshot.

Select **HTML** as the log type in the Logging Preferences Page in FT Java™ or the Logging Options Page in FT .Net.

After playback fails the log opens in your browser.

1. Find the screen snapshot image near the bottom of the log.
    - Click the image or link to view full size.
    - Right click to save, print, or email the JPEG image.

Taking a screen snapshot with scripting

RootTestObject exposes a getScreenSnapshot method that will return a snapshot of the screen. GuiTestObject exposes the same method, but only captures the portion of the screen rendering the TestObject. LogInfo, LogError, and LogWarning all have overloads that will take a snapshot and add it to the log.

Chapter 9. Test Manager Guide

This guide describes how to keep track of the performance of the application by evaluating the test results. Following topics cover how you can work with the test results.

Publishing test result to IBM® Rational® Test Automation ServerHCL OneTest™ Server

The test result indicate the quality of the application under test. There could be different stakeholders to the application who do not have the product installed but want to check the quality of application. You, as the user of the product, can publish the test result to IBM® Rational® Test Automation ServerHCL OneTest™ Server so that others can view it from the web browser.

Before you begin

- You must have installed the Rational® Test Automation ServerHCL OneTest™ Server and started it. The server is part of the product package. For more information about Rational® Test Automation ServerHCL OneTest™ Server, see the Knowledge CenterInformation Center.
- If the IBM® Rational® Test Automation ServerHCL OneTest™ Server server is behind a firewall, you might not be able to connect to it. You must configure the firewall so that it can allow connections on ports 5443 and 7828.

About this task

You can publish only XML and HTML logs. Also, you cannot open Verification Point Comparator from the published result.

Note: This feature is available only with the Eclipse-based IBM® Rational® Functional TesterHCL OneTest™ UI.

To play back Web UI tests and generate reports in Rational® Test Automation ServerHCL OneTest™ Server, you must configure Rational® Test Automation ServerHCL OneTest™ Server from the **Web UI Test** perspective. To play back Functional tests and generate reports in Rational® Test Automation ServerHCL OneTest™ Server, you must configure Rational® Test Automation ServerHCL OneTest™ Server from the **Functional Test** perspective.

You can choose to set the publish parameters once in the Preference page so that you do not have to do it after every run or you can set the parameters every time for the specific result that you want to publish. Based on the publish parameters, the test result is published after the test run. For information about how to publish specific test result, see Publishing specific result to Rational Test Automation ServerHCL OneTest Server.

To set the publish parameters:

1. Click **Window > Preferences > Test > Rational Test Automation Server > HCL OneTest Server > Results**.
2. Type the URL of the server. The format is `https://servername:5443` and click **Test Connection**.

Note: In Linux operating systems, the **Password recovery setup** pop-up window is displayed, but it is hidden behind the **Preferences** dialog and you might not be able to close it. To work around this problem, you must press **ALT+TAB** repeatedly until the **Password recovery setup** pop-up window comes to the foreground.

3. Specify the offline user token that you created on the server. To create a token on Rational® Test Automation ServerHCL OneTest™ Server, see Generating an offline tokenGenerating an offline token.
4. In **Publish result after execution**, select when to publish test result. In the initial stage when you are debugging a test, you might not want to publish the test result. Select one of the options based on your need.
    - Select **Never** to never publish the test results to the server.
    - Select **Prompt** to ask you to publish the test results after every test run.

      Note: A command line run will always publish the test results even if the workbench is set to **Prompt**.

    - Select **Always** to publish the test results after every test execution.
5. In **Publish to Project**, select a project that you are a member of on the server. The test result is published to that project. You cannot create a new project from the desktop client. If there are no projects or you are not a member of any project, create the project or become a member of a project on the server.
6. In **Reports**, select the reports that you want to publish to the server.
7. Click **Apply and Close**.

Results

Test results are automatically published to the Rational® Test Automation ServerHCL OneTest™ Server, depending on the parameters that you have set.

What to do next

To view the test results:

1. Log in to the server.
2. Click the **Results** page and then expand the test.
3. Click the published report links. The report opens in a new browser.

Publishing specific results to the server

If you have a single test result or multiple test results that are not published to IBM® Rational® Test Automation ServerHCL OneTest™ Server, you can publish a single or all of them simultaneously.

Before you begin

- You must have installed the Rational® Test Automation ServerHCL OneTest™ Server and started it. The server is part of the product package. For more information about Rational® Test Automation ServerHCL OneTest™ Server, see the Knowledge CenterInformation Center.
- If the IBM® Rational® Test Automation ServerHCL OneTest™ Server server is behind a firewall, you might not be able to connect to it. You must configure the firewall so that it can allow connections on ports 5443 and 7828.

About this task

You can publish the statistical reports to Rational® Test Automation ServerHCL OneTest™ Server at this moment.

1. Open Rational® Performance TesterHCL OneTest™ Performance, and then go to **File > Export**.
2. Expand the **Test** folder, and then select **Execution Result to Rational Test Automation ServerExecution Result to HCL OneTest Server**.
3. Click **Next**.
4. Expand the project, and then select one or more test results that you want to publish.

   Alternatively, you can right-click on all the test results that you want to publish from **Test Navigator**, and then select **Publish Results**.

   Tip: You can press the Ctrl key to select the results from across projects.

5. Click **New Server** and specify the URL of Rational® Test Automation ServerHCL OneTest™ Server.

   The format of the URL is `https://fully-qualified-domain-name:443`.

   Note: If you had added the publish parameters in the product **Preferences**, then the **Server** field auto-populates the URL of Rational® Test Automation ServerHCL OneTest™ Server.

6. Enter an offline user token that you created on Rational® Test Automation ServerHCL OneTest™ Server.
7. Select a project from the **Project Name** drop-down list.

   The **Project Name** drop-down list displays all the projects on Rational® Test Automation ServerHCL OneTest™ Server.

8. Click a row against the result and type the name of the label in the **Labels** column to add labels to the result.

   Note: If you run the test assets by using the **-labels** option from the command-line interface, then the **Labels** field contains the name of the labels that you provided during the run time and labels that you added to the performance report.

9. **Optional:** Click **Add Common Labels** and type the name of the label to apply a common label to the selected results.

10. Select the reports that you want to publish to the server from the **Reports** section.
11. Click **Publish**.

Results

You have published the test results to Rational® Test Automation ServerHCL OneTest™ Server.

What to do next

To view the test results:

1. Log in to the server.
2. Click the **Results** page and then expand the test.
3. Click the published report links. The report opens in a new browser.

Unified reports

Unified reports in Rational® Functional TesterHCL OneTest™ UI provide a detailed overview of the test results.

You can use the extensive user interface of unified reports to analyze or apply filters to the test results. You can view the unified reports for functional, Windows, mobile, or Web UI tests, immediately after test runs. You can also export unified reports to the desired file formats and use them whenever you require.

To access the reports from an external browser, you can go to **Window > Preferences > Test > Performance Test Reports > Web Reports**, and then select the **Allow remote access from a web browser** checkbox. You can access the reports in one of the following ways:

- **Secured mode**: You can create an authentication layer to access the reports. You can specify a port number in the **Secure port** field of the **Web Reports** dialog, and then provide the login credentials to access the reports.
- **Non-secured mode**: If you want a non-secured access to the reports without any authentication, you can specify a port number in the **Non-secure port** field of the **Web Reports** dialog. You can access the reports without any login credentials.

Note: If you run a test after you select the **Allow remote access from a web browser** checkbox, then the default browser might not display the contents of the unified report. You must restart Rational® Functional TesterHCL OneTest™ UI, and then run the test after you select the checkbox. The unified report is then generated successfully.

You can find more information about unified reports in the following topics:

- Unified reports for Web UI, mobile, and Windows tests on page 1456
- Unified reports for functional test scripts on page 1459
- Prerequisites to export unified reports as a PDF on page 1463
- Customizing PDF files exported from unified reports on page 1464

- Removing customization of PDF files exported from unified reports on page 1467
- Exporting unified reports on page 1468

Unified reports for Web UI, mobile, and Windows tests

Rational® Functional TesterHCL OneTest™ UI generates unified reports for Web UI, mobile, and Windows tests, immediately after test runs.

You can view multiple elements in a unified report. The following table lists the elements with their description:

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>Test Report panel</td><td>Displays the following information:
Total number of tests in numerals
A doughnut chart with green or red rings
The green ring represents the tests that passed, and the red ring represents the tests that failed. Additionally, when you hover the mouse pointer over the green ring, the total number of tests that passed is displayed, and when you hover the mouse pointer over the red ring, the total number of tests that failed is displayed.
A chart legend with green and red colors
The total number of tests that passed is represented by the green color, and the total number of tests that failed is represented by the red color.</td></tr>
<tr><td>Summary tab</td><td>Displays elements as listed in the following table:<table><tr><th>Elements</th><th>Description</th></tr><tr><td>Applications under test</td><td>Displays the names of all the applications under test</td></tr><tr><td>Start time</td><td>Displays the day, date, and time when the test run began</td></tr><tr><td>End time</td><td>Displays the day, date, and time when the test run completed</td></tr><tr><td>Duration</td><td>Displays the duration of the test run</td></tr><tr><td>Tests</td><td>Displays the number of total, passed, and failed tests</td></tr></table></td></tr>
</table>

Element	Description
	<table><tr><th>Elements</th><th>Description</th></tr><tr><td>VP Steps</td><td>Displays the number of total, passed, and failed verification point steps</td></tr></table>
Resource tab	Displays the following information: • Platform You can hover the mouse pointer over the platform's icon (for example, ) to view information about the platform that you used for the test run. • Application runtime You can hover the mouse pointer over the browser's icon (for example, ) to view information about the application runtime environment and the total number of tests in numerals.
Export option	Exports the unified report to different file formats when you select this option. You can save the exported file in the required format and view the test details whenever you require. See Exporting unified reports on page 1468.
Details panel	Displays all the test steps with detailed information.

The **Details** panel of a unified report consists of multiple elements and icons with information about the test steps. The following table provides the list of details that Rational® Functional TesterHCL OneTest™ UI displays in the panel:

Report details	Description	Action
Filtered results	Displays two fields that you can use to filter the passed or failed test results. The filtered results are displayed in the **Test steps** section. One of the following results is displayed and the result depends on the item selected in the first field: • All the tests in a test suite • Only the passed tests in a test suite • Only the failed tests in a test suite One of the following results is displayed and the result depends on the item selected in the second field:	Select an item from the first field to display filtered test suits and select an item from the second field to display filtered test steps.

Report details	Description	Action
	• All the test steps • Only the passed steps • Only the failed steps • Only the passed verification point steps • Only the failed verification point steps	
Test steps	Displays the following details based on the type of test that was run: • **For Web UI, mobile, and Windows tests**: Test steps with details and the corresponding screen captures of steps, if applicable. Steps that failed are highlighted with an exception. • **For compound tests**: Details of the Web UI, mobile, or Windows tests that are contained in the compound test. • **For the AFT Suite**: Details of all the Web UI, mobile, or compound tests that are part of the AFT Suite.	Select a test or test step to view the corresponding details.
View test or step in Rational® Functional TesterHCL OneTest™ UI	Displays the specific test or test step in Rational® Functional TesterHCL OneTest™ UI. The selected test step is highlighted.	Click the  icon.
Type of tests	Displays the type of test that was run as follows: • The  icon is displayed for a single test. • The  icon is displayed for a compound test. • The  icon is displayed for an AFT Suite.	Not applicable
Browser or runtime environment	Displays either of the following icons: • Icon of the browser (for example, ) that was used for web-based application • Icon of Runtime environment  that was used to run the non-web-based application	Hover the mouse pointer over the browser icon to view the browser name and version.

Report details	Description	Action
Screen captures	Displays the screen captures as a slide show.	Click the icon.
Test run information	Displays the test information such as the AUT, host, and duration of the test.	Hover the mouse pointer over the icon.
Expand view of test steps	Displays the complete view of details of test steps.	Click and drag the resize button. **Restriction:** The Internet Explorer browser does not display the resize button.
Failed tests or steps	Displays the number of tests or steps that failed as follows: • The icon is displayed for compound tests. • The icon is displayed for a single test.	Not applicable

Unified reports for functional test scripts

Rational® Functional TesterHCL OneTest™ UI generates unified reports for functional test scripts immediately after test runs. You can view the reports in the default browser window.

You can view multiple elements in a unified report. The following table lists the elements with their description:

Element	Description
Test Report panel	Displays the following information: • Total number of tests in numerals • A doughnut chart with green or red rings The green ring represents the tests that passed, and the red ring represents the tests that failed. Additionally, when you hover the mouse pointer over the green ring, the total number of tests that passed is displayed, and when you hover the mouse pointer over the red ring, the total number of tests that failed is displayed.

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td></td><td>
A chart legend with green and red colors
The total number of tests that passed is represented by the green color, and the total number of tests that failed is represented by the red color.
</td></tr>
<tr><td>Summary tab</td><td>
Displays elements as listed in the following table:
<table>
<tr><th>Elements</th><th>Description</th></tr>
<tr><td>Applications under test</td><td>Displays the names of all the applications under test</td></tr>
<tr><td>Start time</td><td>Displays the day, date, and time when the test run began</td></tr>
<tr><td>End time</td><td>Displays the day, date, and time when the test run completed</td></tr>
<tr><td>Duration</td><td>Displays the duration of the test run</td></tr>
<tr><td>Tests</td><td>Displays the number of total, passed, and failed tests</td></tr>
<tr><td>VP Steps</td><td>Displays the number of total, passed, and failed verification point steps</td></tr>
</table>
</td></tr>
<tr><td>Resource tab</td><td>
Displays the following information:
Platform
You can hover the mouse pointer over the platform's icon (for example, ) to view information about the platform that you used for the test run.
Application runtime
You can hover the mouse pointer over the browser's icon (for example, ) to view information about the application runtime environment and the total number of tests in numerals.
</td></tr>
<tr><td>Export option</td><td>Exports the unified report to different file formats when you select this option. You can save the exported file in the required format and view the test details whenever you re-quire. See Exporting unified reports on page 1468.</td></tr>
</table>

Element	Description
Details panel	Displays all the test steps with detailed information.

The **Details** panel of a unified report consists of multiple elements and icons with information about the test steps. The following table provides the list of details that Rational® Functional TesterHCL OneTest™ UI displays in the panel:

Report details	Description	Action
Filtered results	Displays two fields that you can use to filter the passed or failed test results. The filtered results are displayed in the **Test steps** section. One of the following results is displayed and the result depends on the item selected in the first field: • All the tests in a test suite • Only the passed tests in a test suite • Only the failed tests in a test suite One of the following results is displayed and the result depends on the item selected in the second field: • All the test steps • Only the passed steps • Only the failed steps • Only the passed verification point steps • Only the failed verification point steps	Select an item from the first field to display filtered test suits and select an item from the second field to display filtered test steps.
Test steps	Displays test steps with details and the corresponding screen captures of steps, if applicable. Steps that failed are highlighted with an exception.	Select a test or test step to view the corresponding details.
View test or step in Rational® Functional TesterHCL OneTest™ UI	Displays the specific test script or test step in Rational® Functional TesterHCL OneTest™ UI. The line number of the test script is highlighted.	Click the  icon.
Browser or runtime environment	Displays either of the following icons:	Hover the mouse pointer over the browser icon to view the browser name and version.

Report details	Description	Action
	• Icon of the browser (for example,) that was used to run HTML tests • Icon of Runtime environment that was used to run non-HTML tests	
Screen captures	Displays the screen captures as a slide show.	Click the icon.
Test run information	Displays the test information such as the AUT, host, and duration of the test.	Hover the mouse pointer over the icon.
Expand view of test steps	Displays the complete view of details of test steps.	Click and drag the resize button . **Restriction:** The Internet Explorer browser does not display the resize button.
Failed tests or steps	Displays the 11 icon for the number of steps that failed in a single test.	Not applicable

Additional information in Unified reports for HTML applications

You can view additional information in a unified report after a functional test run, if you set preferences in Rational® Functional TesterHCL OneTest™ UI before you run the test.

Note: This feature is applicable only when you run the functional tests for HTML applications by using the Web UI engine.

You can set the following preferences in Rational® Functional TesterHCL OneTest™ UI before you run the test:

- To set the user profile, you can go to **Window > Preferences > Test > Test execution > UI Test Playback > Browser**, and then select the **Browser User Profile** checkbox.

  The profile path is displayed in the field. The checkbox is not selected as the default option.

- To clear cache or browser history, you can go to **Window > Preferences > Test > Test execution > UI Test Playback > Desktop**, and then select the **Clear cache** or **Clear history** checkboxes.

The checkboxes are not selected as the default options.

- To delete cookies, you can add `deleteCookies()` in the test script.

If you configured these preferences earlier and do not want to view the additional information for the current test, you can change the preferences and then run the test.

After you set the preferences in Rational® Functional TesterHCL OneTest™ UI and run the test, the additional information that you can view in the unified report are listed in the following table:

Additional information	Location in the unified report
Profile name	The **Start application** section of the **Details** panel. The profile path is displayed.
Clear cache	The **Start application** section of the **Details** panel. Clear cache is displayed as either *False* (default option) or *True*.
Clear history	The **Start application** section of the **Details** panel. Clear history is displayed as either *False* (default option) or *True*.
Delete cookies	The **Details** panel. The script about the Delete cookies information is displayed as a step.

Prerequisites to export unified reports as a PDF

After you run tests in Rational® Functional TesterHCL OneTest™ UI, the results are displayed in unified reports. Before you export the unified reports as a PDF, you must have read about the prerequisite tasks that you can perform to customize the appearance of the PDF files that are exported from the unified reports.

You can perform the following customizations in the exported PDF files:

- Highlight only the verification points and exclude the details displayed in the tabular form for test steps in the PDF.
- Increase the size of screen captures that are displayed in the PDF.
- Exclude screen captures of specific SAP GUI test steps in the PDF.

Note: If you do not customize the preferences for the exported PDF, the PDF displays the details of the test steps in the tabular form along with images of the screen captures in their default size. This default format

might render the PDF to run into several pages and the details of the verification points might confuse you to correlate with the images.

For information about the tasks that you must perform to customize the view of the exported PDF, see Customizing PDF files exported from unified reports on page 1464.

You can also perform the customizations from the command line by using the **-vmargs** option. For more information, see Configuration of test runs from the command line on page 1395.

If you have already customized the view of the exported PDF files and you want to remove the customization, see Removing customization of PDF files exported from unified reports on page 1467.

Languages supported for PDF export

When you set up your computer on which you installed Rational® Functional TesterHCL OneTest™ UI in any of the supported languages, you can also view the PDF files that you export from the unified reports in any of the following supported languages:

Supported language to export PDF files	Locale set in your computer
Brazilian Portuguese	pt_BR
English	en_US
French	fr_FR
German	de_DE
Hungarian	hu_HU
Italian	It
Spanish	es

Customizing PDF files exported from unified reports

When you want to change the default view of PDF files that are exported from unified reports, you can customize the view of the exported PDF files.

Before you begin

You must have run tests in IBM® Rational® Functional TesterHCL OneTest™ UI that have generated the unified reports.

About this task

You can open the `eclipse.ini` file and add the required parameters with their values in the file to customize the PDF when you export a unified report. After you save the `eclipse.ini` file, you must then restart IBM® Rational® Functional TesterHCL OneTest™ UI before you export unified reports.

1. Open the `eclipse.ini` file that is located in the `C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest` folder.
2. Determine the parameters that you want to set as a preference in the `eclipse.ini file` from the parameters listed in the following table:

Parameter	Supported values	Action
`-Dreport.skipTableInPdf`	• `false` • `true`	Perform any of the following actions: • Enter this parameter with the value set to `false`, if you already set the value to `true` and you want to revert to the default view. When the value is set to `false`, the verification points in the test steps are not highlighted and details are displayed in a tabular form in the exported PDF. For example, enter the following parameter to view details of test steps in the tabular form, without highlighting the verification points: `-Dreport.skipTableInPdf=false`. **Note:** This setting is the default preference for the exported PDF files for unified reports in IBM® Rational® Functional TesterHCL OneTest™ UI. • Enter this parameter with the value set to `true`. When the value is set to `true`, the verification points in the test steps are highlighted and the step details in the tabular form are not displayed in the exported PDF. For example, enter the following parameter to view test steps where the verification points are highlighted but the step details in the tabular form are not displayed: `-Dreport.skipTableInPdf=true`.
`-Dreport.snapShotSize`	Size of images relative to the page-width	Enter a value between 50 and 100 so that the images are increased to fit within the page margins in the exported PDF. For example, enter the following parameter to increase the width of the images to fit within 75% of the page width while the height is increased proportionately: `-Dreport.snapShotSize=75`. **Note:**

Parameter	Supported values	Action
		• If the parameter value is empty, or if you did not add this parameter in the `eclipse.ini` file, IBM® Rational® Functional TesterHCL OneTest™ UI automatically sets the size of images to 50% of the page width as the default size. • You can use this parameter, only when you have set `-Dreport.skipTableInPdf=true`.
`-Dreport.skipActionEventImageList`	`[objectType]:[methodName]`	Enter the name of the objects in the SAP GUI application along with the name of the methods. You can use the pipe character `|` as the delimiter between different objects. The corresponding screen captures are excluded in the generated PDF. **Note:** • This parameter is applicable for SAP GUI applications only. • You can use this parameter only if you have installed IBM® Rational® Performance TesterHCL OneTest™ Performance with IBM® Rational® Functional TesterHCL OneTest™ UI in the shell-shared mode. • You can use this parameter only when you set `-Dreport.skipTableInPdf=true`. For example, enter the following values for the parameter to exclude images captured for the `maximize` and `setTest` methods of the `GuiMainWindow` and `GuiCheckBox` objects, respectively: `-Dreport.skipActionEventImageList=GuiMainWindow:maximize|GuiCheckBox:setTest`

3. Save the `eclipse.ini` file.
4. Restart IBM® Rational® Functional TesterHCL OneTest™ UI with the `-clean` option.

Results

You have successfully customized the view of the exported PDF from unified reports.

What to do next

You can view and export the unified reports. See Exporting unified reports on page 1468.

Related information

Prerequisites to export unified reports as a PDF on page 1463

Configuration of test runs from the command line on page 1395

Removing customization of PDF files exported from unified reports on page 1467

Removing customization of PDF files exported from unified reports

When you customized the view of the PDF files exported from unified reports and you want to revert to the default view, you can remove the customization anytime.

Before you begin

You must have completed the following tasks:

- Customized the view in PDF files exported from unified reports. See Customizing PDF files exported from unified reports on page 1464.
- Exported PDF files from unified reports. See Exporting unified reports on page 1468.

About this task

You can open the `eclipse.ini` file and remove the parameters with the values that you set previously to customize the view of the exported PDF files.

After you save the `eclipse.ini` file, you must then restart IBM® Rational® Functional TesterHCL OneTest™ UI before you export PDF files in the default view from unified reports.

1. Open the `eclipse.ini` file that is located in the `C:\Program Files\IBM\SDPC:\Program Files\HCL\HCLOneTest` folder.
2. Determine the parameters in the `eclipse.ini` that correspond to the customization that you no longer require. The parameters that you had added earlier for customization can be any of the following:
    - `-Dreport.skipTableInPdf`

      The parameter is used to highlight verification points in test steps and hide the tabular details of step steps from the generated PDF.

    - `-Dreport.snapShotSize`

      The parameter is used to increase the size of screen captures, relative to the page width, that is displayed in the PDF.

    - `-Dreport.skipActionEventImageList`

      The parameter is used to exclude screen captures of only specific SAP GUI test steps in the PDF.

Note: If you want to remove either the `-Dreport.snapShotSize` or `-Dreport.skipActionEventImageList` parameter with the values, you must also remove the `-Dreport.skipTableInPdf` parameter with the values.

3. Delete the parameters that you no longer require with their values from the `eclipse.ini` file.
4. Save the `eclipse.ini` file.
5. Restart IBM® Rational® Functional TesterHCL OneTest™ UI with the `-clean` option.

Results

You have successfully removed the customization to view the exported PDF files that you no longer require.

What to do next

You can view and export the unified reports. See Exporting unified reports on page 1468.

Related information

Prerequisites to export unified reports as a PDF on page 1463

Configuration of test runs from the command line on page 1395

Exporting unified reports

After you run tests and when the unified report is generated, you can export the unified report to different file formats such as HTML, PDF, or XML(JUnit). By using the exported file, you can view the test details in the desired format and save the exported file for later use.

About this task

On the unified report page, you can click the **Export** button at the top-right corner to directly export the unified report to an HTML file, which is the default option. The exported report includes the summary, test steps, and screenshots.

You can also export the unified report to other available file formats and set preferences for the details that you want to export.

1. Click the ellipses button  .
2. Select one of the following file formats from the **Choose File Format** list:
   - **HTML(Default)**: Select **HTML(Default)** to export the report in an HTML file.

     Note: The HTML file is downloaded as a zip file. You must extract the zip file to view the HTML file.

   - **PDF**: Select **PDF** to export the report in a PDF file.

Restriction:

- The verification points that are captured for functional tests are not included in the PDF file.
- The icons of the applications under test are not supported in the PDF file.
- The text in a table that contains the greater than symbol **>** is printed as `&gt` in the generated PDF of the unified report.

- **XML(JUnit)**: Select **XML(JUnit)** to export the report in the JUnit format. You can use the JUnit XML to publish test results to Continuous integration/continuous delivery (CI/CD) pipelines. For example, Azure Pipeline.

3. Select one of the following checkboxes to specify the details that must be included in the exported file.
   - **Include Steps**: Select this checkbox to include the summary and steps of the playback in the report.
   - **Include Screenshots**: Select this checkbox to include the summary, test steps, and screenshots of the playback in the report.

   Notes:

   - When both the checkboxes are not selected, then the exported report contains only the summary.
   - These checkboxes are disabled when you select **XML(JUnit)** because they are not applicable for the JUnit format.

4. Click **Export**.

Note: If you encountered any errors after you clicked **Export** that indicate insufficient disk space or memory, you must increase the default memory allocated in the eclipse configuration file `eclipse.ini` that is located in your eclipse installation directory. Open the file, find the *-Xmx4000m* or *-Xmx4g* attribute, and then change it to *-Xmx6000m* or *-Xmx6g*.

Results

You exported the unified report in the selected format.

Results for tests in UI Test perspective

In this section, you will learn how to analyze test results in the Web UI Test perspective.

UI Test Statistical report

You can use the UI Test Statistical report to view the performance data for your application. This report displays performance data for all of the transactions in the test, all of the steps in the test, and all of the supported computer resources.

The UI Test Statistical report provides you with the following information:

<table>
<tr><th>Page</th><th>Description</th></tr>
<tr><td>Overall</td><td>Displays the high-level data such as the name of the test, total time taken to run the test, the number of steps attempted, and the number of steps that passed or failed.</td></tr>
<tr><td>Transaction Performance</td><td>Displays the Net End-to-End time and Net Server time for all of the transactions in the test.</td></tr>
<tr><td>Step Performance</td><td>Displays the Net End-to-End Time, Net Server Time, and On App Time for all of the steps in the test. You must scroll down the page to view the response times for all of the browsers that were used for the run.</td></tr>
<tr><td>Resources</td><td>Displays graphically the metrics collected of the mobile device resources and resources used by the mobile applications for Android or iOS mobile tests.

You can find the following key Resource Monitoring parameters that are supported for Android or iOS mobile tests:
<table>
<tr><th>Parameter</th><th>Android tests</th><th>iOS tests</th><th>Description</th></tr>
<tr><td>Application CPU%</td><td>Yes</td><td>Yes</td><td>Displays the CPU utilization by the application that was tested during the test run.</td></tr>
<tr><td>Device CPU%</td><td>Yes</td><td>No</td><td>Displays the CPU utilization of the device on which the tests were run for the duration of the test run.

Note: A lag is observed in the metrics reported for the initial part of the test run.</td></tr>
<tr><td>Application Used Physical Memory</td><td>Yes</td><td>Yes</td><td>Displays the utilization of memory for the application that was tested during the test run.</td></tr>
<tr><td>Device Battery%</td><td>Yes</td><td>Yes</td><td>Displays the power utilization by the device battery during the test run.</td></tr>
</table></td></tr>
</table>

In a UI Test Statistical report, you can sort the order of the test steps, which are listed in the **Step** pane, either by alphabetical order or by the order of execution of the test steps. The default sorting of the test steps is by the order of execution.

You can either click the **Execution order** icon or the **Alphabetical order** icon to toggle between the sorting options. You can also click the Up Arrow icon to sort the test steps in the correct order of execution or the correct alphabetical order. Similarly, you can click the Down Arrow icon to sort the test steps in the reverse order of execution or the reverse alphabetical order.

Evaluating desktop Web UI results

After the test run, Unified Report the statistical report, and test log is displayed. The statistical report displays response time data for transactions and test steps. The live report displays result for the functionality of each step and the test log shows details of each test step.

You can customize and export the reports.

Customizing reports

To view reports in the manner that you want, you can customize a report. For information about the options available to customize a report, see Customizing reports on page 1479.

Exporting reports

To create executive summary and export reports in HTML and CVS format, see Exporting data from runs on page 1488.

UI Test live report

When you initiate a test run, the **Test Execution** perspective displays the UI Test report. This report shows the status of each step that is being currently running on the browser. After the test run completes, the UI Test Report shows up in a new tab.

You must complete the following steps if you want the UI Test live report to be generated:

1. Open the `eclipse.ini` file that is located in the path as: `/opt/IBM/SDP/eclipse.ini /opt/HCL/HCLOneTest/eclipse.ini`.
2. Add the following property:

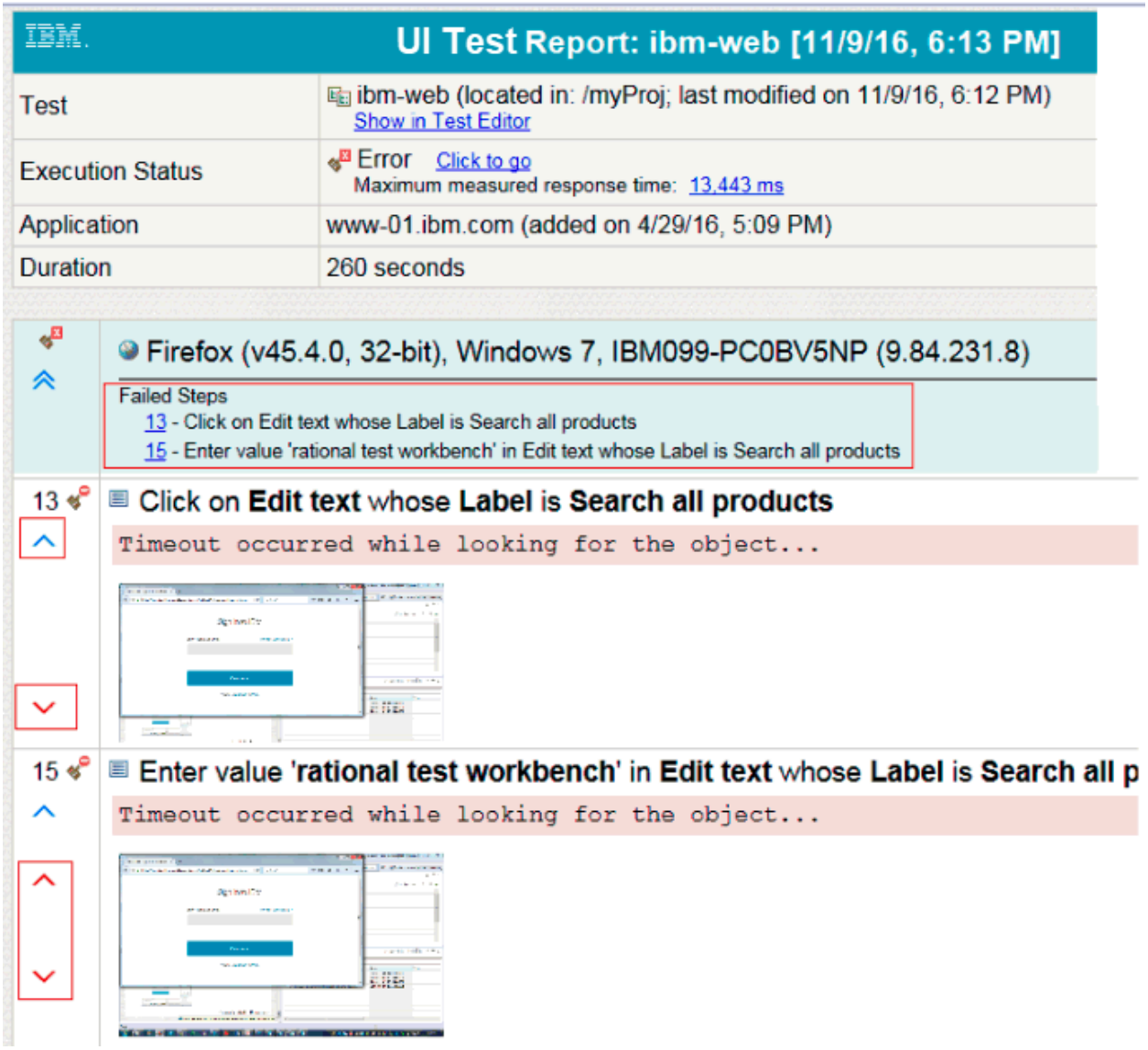
   -DdisableMoebReport=false

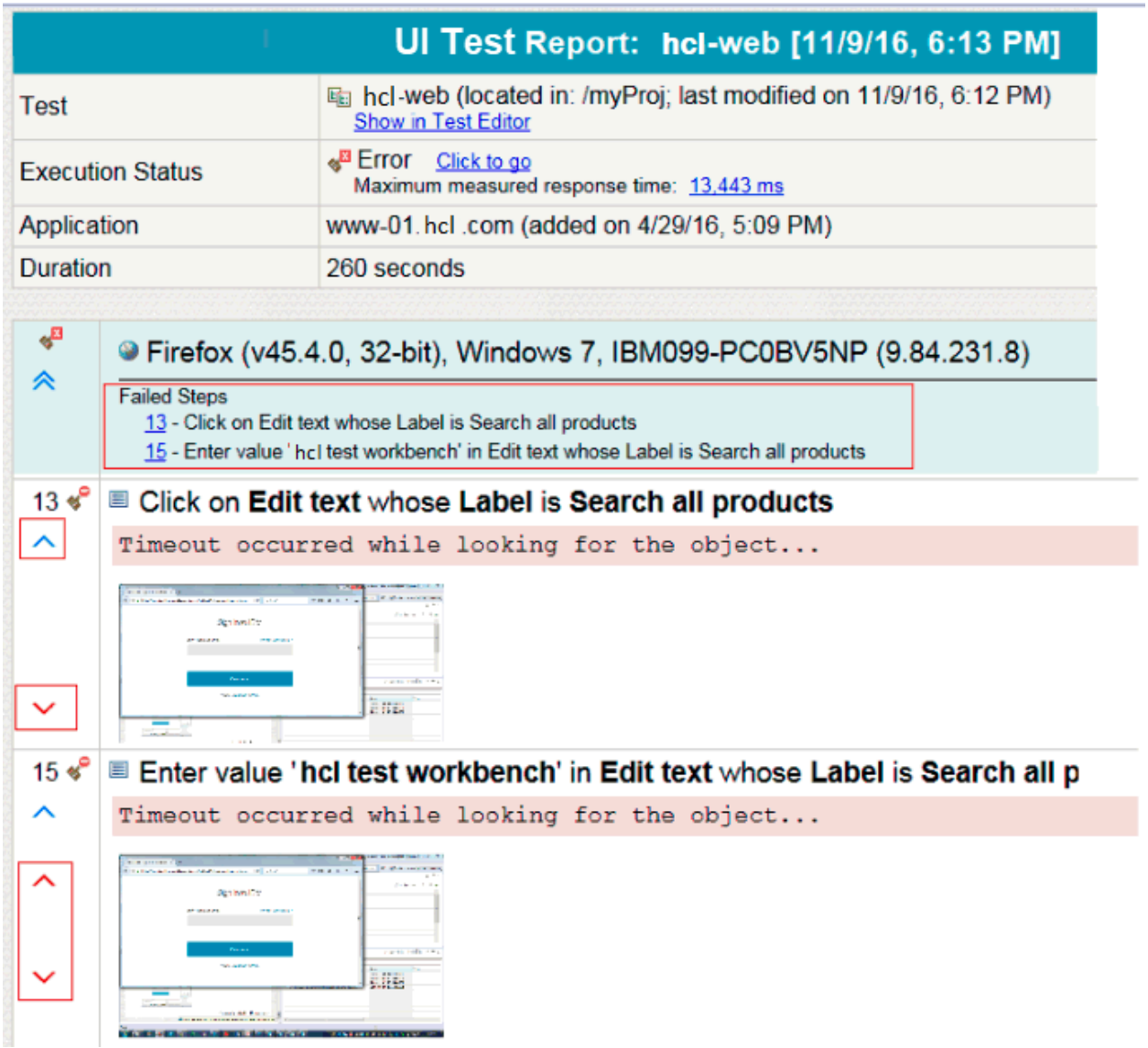
3. Save and close the `eclipse.ini` file.
4. Restart Rational® Functional TesterHCL OneTest™ UI.

When you run Web UI or mobile tests, the UI Test Live report is generated and stored in the **Results** folder.

If a test script includes many steps or if a test is run on multiple browsers simultaneously, navigating the errors in the UI Test report is cumbersome.

The test header shows the browser name and version the test was run and the steps that failed against the browser. You can also navigate from one error to another by clicking the red color arrow shaped icons. The blue color arrow shaped icons are used to reach to the header of the test.





If you chose to run the test on multiple browsers in parallel, the report lists each browser on which the test was run.

Test	ibm (located in: /myProj; last modified on 2/9/17, 11:37 AM) Show in Test Editor
Execution Status	Failure Click to go Maximum measured response time: 20,159 ms
Application	www-01.ibm.com (added on 6/27/16, 11:29 AM) 1 2
Browser	Chrome (v56.0.2924.87, 64-bit), Windows 7, IBM099-PC0BV5NP (9.77.192.251) 1
Browser	Firefox (v45.7.0, 32-bit), Windows 7, IBM099-PC0BV5NP (9.77.192.251) 2
Duration	95 seconds

Generating Functional Test Report

To create an HTML report that can be printed or used outside of the workbench, generate a functional test report. The functional test report shows the verdicts for the test and for each step of the test. Functional reports are generated from the test run as HTML files that use predefined report designs.

To generate a report:

1. In the Test navigator view, right-click a test run and click **Generate Functional Test Report**.
2. Select a project to store the report, specify a name, and click **Next**.
3. Select one of the following report templates and click **Finish**:
    a. To generate a report for Web UI tests, click **Web UI reports (XSL)**.
    b. To generate a report for Selenium tests, click **Selenium reports (XSL)**.
    c. To generate a report for a IBM® Rational® Functional TesterHCL OneTest™ UI test that is part of a compound test, click **Functional GUI reports (XSL)**.

Results

The report is displayed in the new tab. If you close the tab, you have to generate the report again.

Low Intensity Performance Testing

In performance testing, the user load is gradually increased from low intensity to high intensity. Testing the performance of the application with very few users is referred to as Low Intensity Performance Testing. Typically, you require performance test assets to do performance testing. However, in IBM® Rational® Functional TesterHCL OneTest™ UI, you can do functional testing and low intensity performance testing with the same functional test assets.

Doing low intensity performance testing helps you find performance bottlenecks in the initial phase of development itself thereby reducing the cost and effort that would be required when the performance issue is discovered in the later phases of the release.

As part of low intensity performance testing, you can measure the various response times such as Net End-to-End time, Net Server Time, On App and Off App response time for the transactions and each UI action. The UI Test Statistical report displays all of these response times.

- Net End-to-End time : It is a measured time of interactions with the server and a client such as a browser or a device. Time taken over the network is also taken into account. Typically, this does not include think time or processing time by the workbench.
- Net Server time : It is a sum of the time of interactions with the server and the network. This time does not include client (browser or device) time, think time or processing time by the workbench.
- On App time - It is a measured time of interactions on the client (browser or device) itself.
- Off App time - It is a sum of the time of interactions with the server and the network. The time spent on the client is not taken into account.

A transaction is a logical grouping of a few UI actions. For example, creating a user is a transaction and filling up each user field to create a user is a UI action. For an end user, time taken to complete a business task (transaction) is more useful than time taken for each UI action. Over time certain business tasks get higher priority to be measured for performance improvement. The Transaction report displays information about all of the transactions in a test.

Using a transaction might give you data about the response time taken by a transaction. However, it would be more useful if you knew that that response time for a transaction is at the expected level. The end users would have already defined response times for each business task in the service level agreement. So, you can define those performance requirements in the tool and check whether the measured response time adheres to the agreement. The Performance Requirement report displays information whether the transactions adheres to the performance criteria.

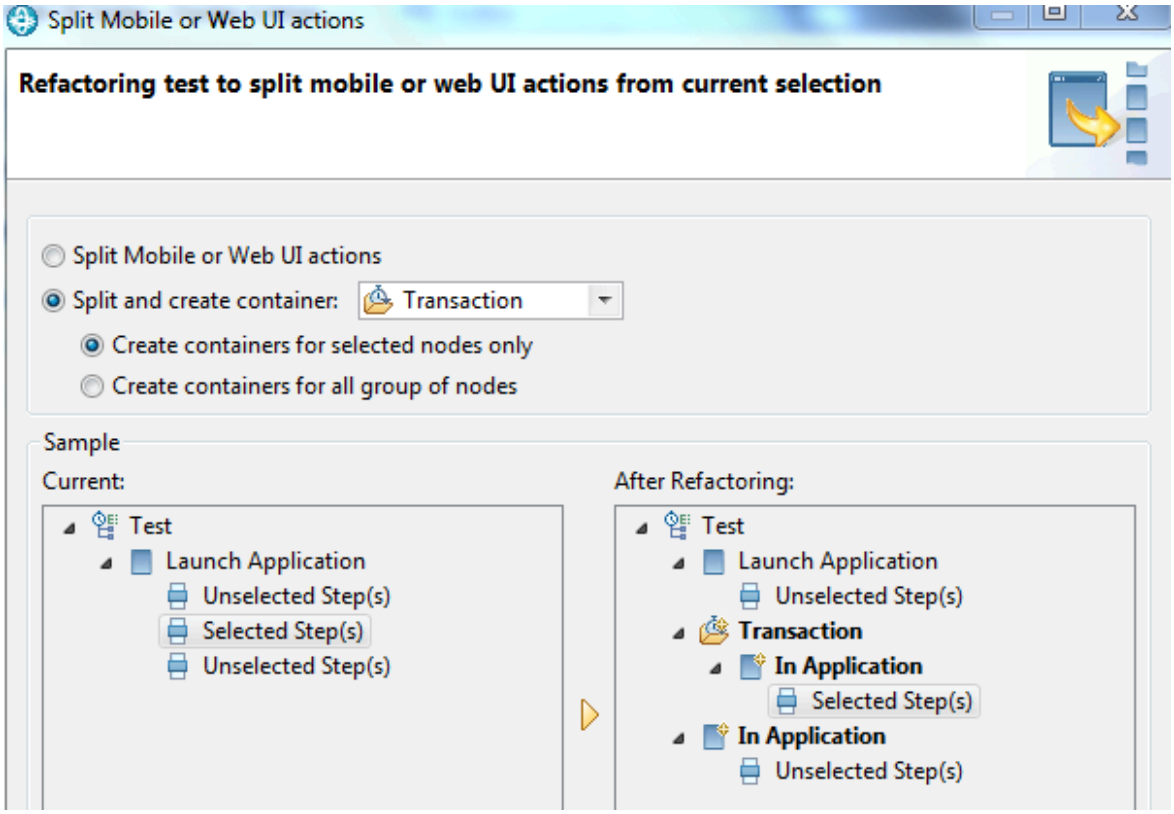
Adding a transaction

A transaction is a business scenario in the application under test such as logging in, checking out a product, or making a payment. It is a logical grouping of certain UI actions. You add a transaction to the test to check the performance for the entire transaction rather than each UI action in the transaction.

About this task

You can add a transaction only for the Launch App or In App nodes. So, if you have a bunch of steps that justifies a transaction, split them from the rest of the steps to add a transaction.

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. In the test editor, select one or more steps in the test script for splitting into one or more application nodes.
3. Right-click the selected steps, and then select **Split Mobile actions or Web UI actions**.
4. In the refactoring dialog box that opens, click **Split and create containers**. Ensure that **Transaction** is selected.



5. To add a transaction only for the selected steps, select the **Create containers for selected nodes only** option.
6. To add a transaction for all the nodes (Launch App and In App) in the test, select the **Create containers for all group of nodes** option.
7. Click **Finish**.

   Result

   The group of steps are nested in the In App node which is also nested in a Transaction.

What to do next

After the test run completes, to view the Transaction report, in the Statistical report tab, select **Transaction Report**. To view information about each report, click the help icon.

Defining performance requirements in transactions

You can define performance requirements for transactions in a test. These requirements specify acceptable thresholds of performance for transactions and validate service level agreements.

Before you begin

Add a transaction to the test. See Adding a transaction on page 1475.

About this task

For example, you might define that the login action should take between 1-2 seconds and the check-out action should take between 4-5 seconds. If the response time is outside of these threshold values, the Performance Requirements

report will display the results. You define a performance requirement as standard or supplemental. A standard performance requirement is a requirement that you determine as significant enough to cause the entire run to be declared a failure if it fails. A supplemental performance requirement, although important, is not significant enough to cause the run to fail.

1. Select a transaction in the test and in the **Transaction Details** area, click the **Advance** tab.
2. Select the **Enable Performance Requirements** check box.
3. Click the performance requirement to define, and add a definition, as follows:

Option	Description
Name	You can change the name of a performance requirement to improve readability. However, changing a requirement name causes a mismatch between the Performance Requirements report, which uses the changed name, and the other reports, which use the default name. Therefore, when you change a requirement name, be sure to keep track of the original name.
Operator	Click to select an operator.
Value	Type a value.
Standard	Select to make the requirement standard. A standard requirement can cause a test to have a verdict of fail. Clear to make the requirement supplemental. In general, supplemental requirements are used for requirements that are tracked internally. A supplemental requirement cannot cause a run to fail, and supplemental results are restricted to two pages of the Performance Requirements report.

4. Optionally, apply the requirements to the other transactions in the test:
    a. In the Requirements table, right-click the requirement row, and select **Copy Requirements**.
    b. Select the transaction to apply requirements, and in the Requirements table right-click a requirement row and select **Apply Requirements**.
5. Optionally, to remove the definitions of a requirement, right-click a requirement and click **Clear**.

What to do next

Now, run the test and see the Performance Requirements report.

Viewing On App and Off App response time

When you initiate a test run from the test workbench, by default, the response time is collected and displayed in the UI Test Report after the run. This report shows the overall response time for each test step. The response time is also displayed on the **Step Performance** tab of the statistical report. This report shows the average response time for each test step.

About this task

If you used the Mozilla Firefox or Google Chrome browser to run the test, you can drill down further and see the response time taken at each step by the application (On App), and the server and network (Off App).

To view the On App and Off App data for all of the steps on a page, from the Step Performance page, click the > symbol after the Step Performance header and then click **Step Response Time Contributions**.



1. Click **Run Test** in the test workbench.
2. In the **Run Configuration** wizard, select the Firefox or Chrome browser and click **Run**.
   When the test is completed, the statistical report displays graphs that represent the response time for each step in the test. By default, only the 10 highest response times are collected during playback. You can modify the filter by using the context menu on the legacy report and by going to the edit mode on the web report.
3. Click any response time graph and select **Step Response Time Contributions**. The report displays two bars, one to show the time spent by the step on the application (**On App Time**) and another to show the time spent on the network and server (**Off App Time**).

   Note: The **Step Response Time Contributions** option is available only if you played back the test with the Firefox or Chrome browser.

   Figure displaying the Step Performance tab with Step Response Time Contributions context menu.

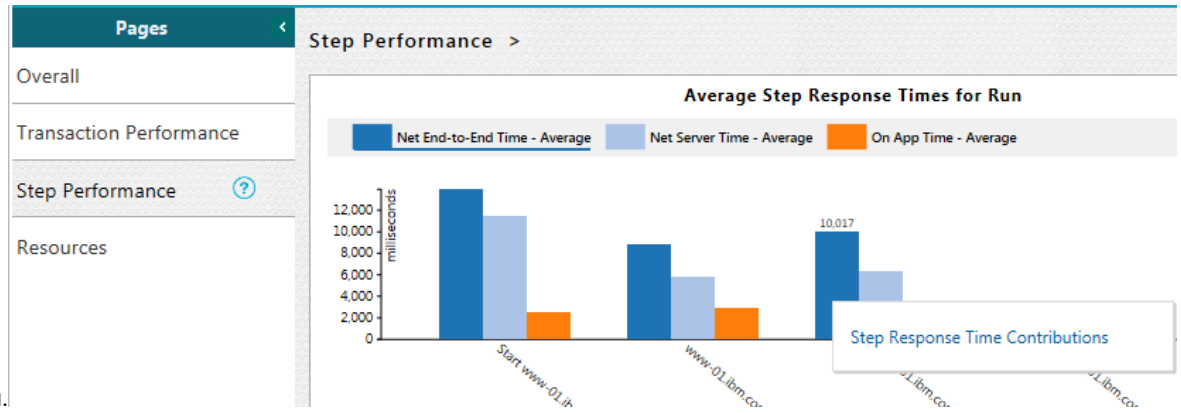
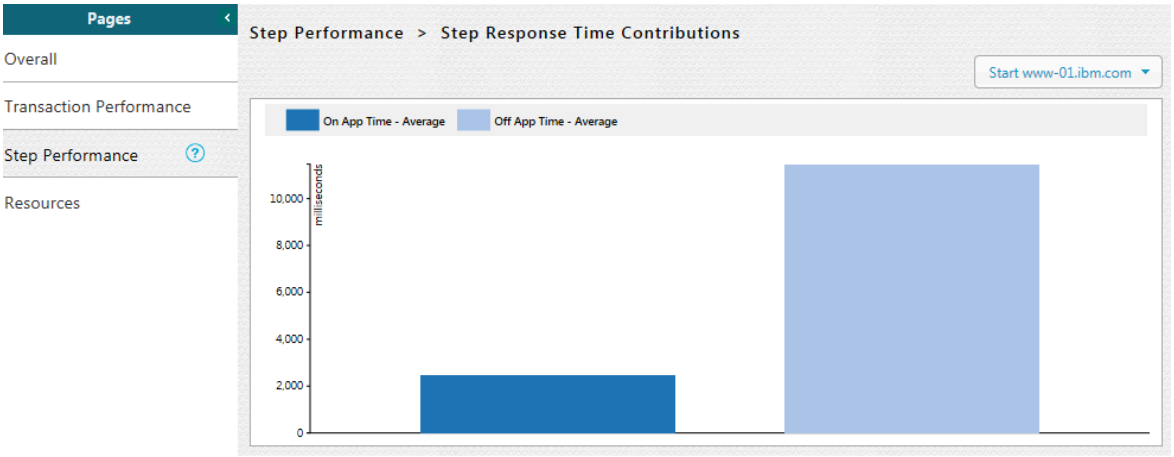


Figure displaying On App Time and Off App Time.



4. To return to the main page, click the **Step Performance** link at the top of the **Step Performance** tab.

Customizing reports

You can customize reports to specifically investigate a performance problem in more detail than what is provided in the default reports.

Creating custom reports

If the default reports do not address your needs, you can create your own reports.

About this task

Before you create a custom report, determine the ways in which the custom report will be different from or similar to the system-supplied reports. You can use a default report as a template, modify the counters, and save it with a different name. You can create a copy of pages or charts in a report that are based out of existing pages or charts. To copy the pages or charts, go to the Edit view and click the **Duplicate** icon.

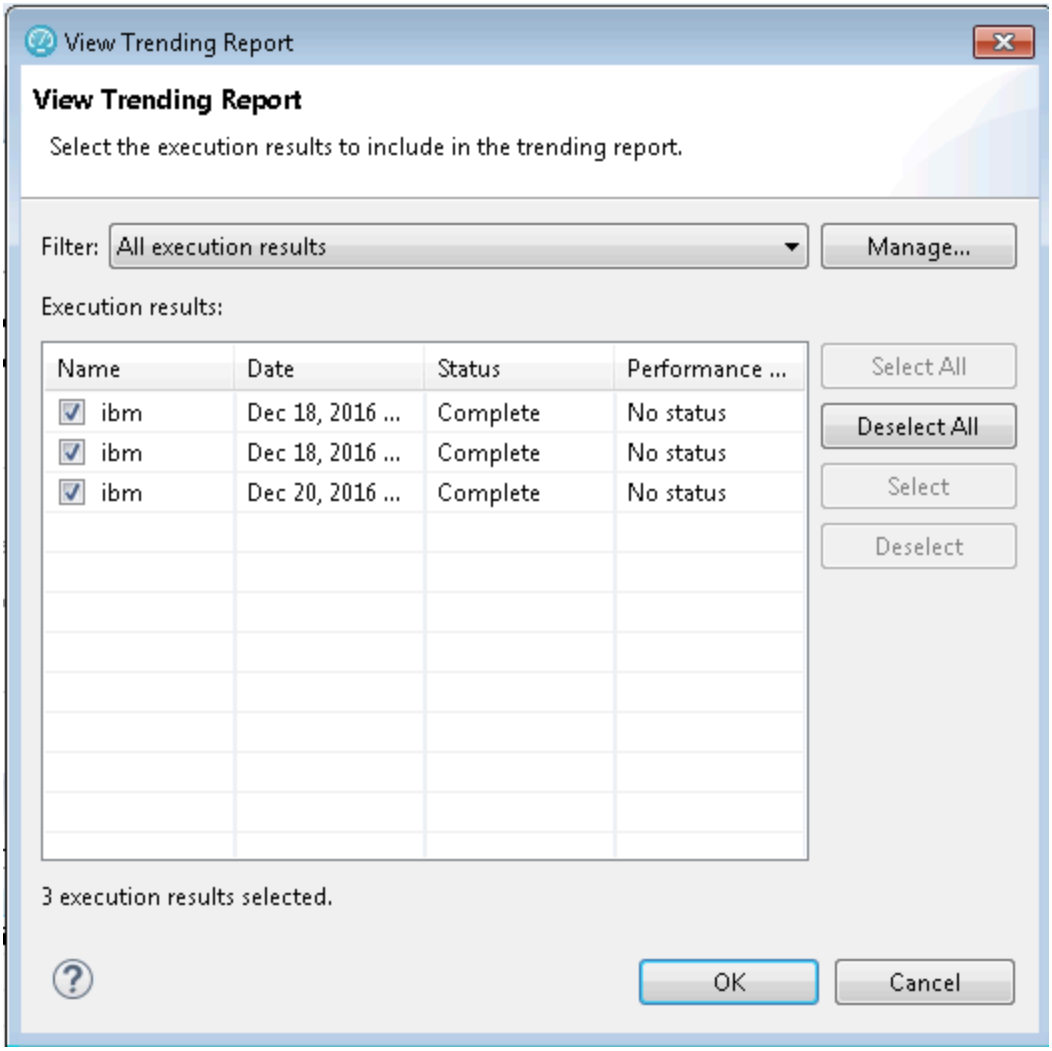
1. From the report, click **Menu** and click **New**.
2. In **Create a new report** dialog box, specify a name and description about the new report and click **Create**.
3. To change the page title, click the default page title and specify a different name.
4. Click **Click to insert a row** and specify the number of columns to add the views.
   Each view represents a bar chart, line chart, or pie chart.
5. Select a view. To add counters to the view, click **Settings**.
6. On the View Settings page, select a counter and add its details.
7. Click **Apply** and from the **Menu**, click **Save** to save the report.
8. To add more views to the report, repeat steps 4 through 7 again.

Viewing trending reports

To view the trend of response time for an application over a period of time, open the trend report for a run. In addition to the response time, you can view the trend for the loops, transactions, and performance requirements for the application.

About this task

The trend report can help you determine the response times of the application at different milestones. For instance, you can run the performance test for sprint or milestone builds and tag them. When generating the trend report, you can specify conditions such as results that are less than 60 days old and include 'milestone' tag.



You cannot save a trend report. So, if you close the report, you have to generate it again.

To view the trend report:

1. In the Test Navigator view, select the run for which to open the trend report.
2. Right-click the run and click **View Trend Report**.

3. To view the trend that is based on certain criteria, in **Filter**, select a filter criteria.
   If there is no customized filtering criteria, create one by clicking **Manage** and then **Add**.
4. To save the criteria, click **Save**, specify a name to the filter, and click **OK**.
   The results in the execution results table are filtered out according to the specified criteria.
5. Click **OK**.
   Result
   The trend report is generated.

Filtering data in test results

You can filter the data in a test result that is displayed in a report so that you can remove the unnecessary data and focus on the data that is significant for the analysis.

Before you begin

You must have a test result.

1. Double-click the test result from the **Test Navigator**.
2. Select a report from the drop-down list.
   For example, the Performance Report.
3. Click the **Menu** icon  , and then click **Edit**.
4. Select a page from the left pane in which you want to filter the data.
   For example, the Page Performance page.
5. Click the **Settings** icon  on a specific graph or table.
6. Click the **Filters** tab on the **View Settings** page.
7. Perform any of the following actions described in the following table to filter the data:

Options	Actions
Filter by count	Perform the following steps: a. Clear the **Show highest values** check box to display the smallest values for the pages. **Note:** By default, the **Show highest values** check box is selected. b. Enter a value in the **Number to display** field to display the items on the graph or table based on the specified value for the selected counter. **Note:** The title of the page is updated with the value that you specified along with the **Show highest values** field.

Options	Actions
	For example, if you selected the **Show highest values** check box and entered 10 in the **Number to display** field for the Performance Summary page, then the title is displayed as follows: **Performance Summary (10 Highest)**. c. Select the counter from the **Primary counter for table filtering** field by using the drop-down list if you want to filter the data for the other counter. d. Select the component for the counter that you selected from the **Component** drop-down list. For example, consider that you performed the following actions to filter the data: ◦ Selected the **Show highest values** check box. ◦ Entered *5* as a value in the **Number to display** field. ◦ Selected *Page Response Time* as **Primary counter for table filtering** and *Average* as **Component**. Then, the graph or table displays *5* pages that include the highest *Average Page Response Time* during the test run.
Filter by value	Perform the following steps: a. Clear the **Show counters above value** check box to display the lower values for the pages. **Note:** By default, the **Show counters above value** check box is selected. b. Enter a value in the **Filter value** field to display the items on the graph or table based on the specified value for the selected counter. c. Select the counter from the **Primary counter for table filtering** field by using the drop-down list if you want to filter the data for the other counter. d. Select the component for the counter that you selected from the **Component** drop-down list. For example, consider that you performed the following actions to filter the data: ◦ Cleared the **Show counters above** value check box. ◦ Entered *800* as a value in the **Filter value** field. ◦ Selected *Page Response Time* as **Primary counter for table filtering** and *Average* as **Component**.

Options	Actions
	Then, the graph or table displays the pages that include the *Average Page Response Time* lesser than *800 ms* during the test run.
Filter by name	Perform the following steps: a. Enter a label name in the **Filter value** field. The label name is the name that you provided for a page when you recorded the test. b. Select the **Case sensitive** check box to find the pages that exactly match with the letter case of the name that you entered in the **Filter value** field. c. Select any of the following options to find pages more effectively: ▪ Include counters whose label contains filter value ▪ Include counters whose label equals filter value ▪ Exclude counters whose label contains filter value ▪ Exclude counters whose label equals filter value

Note: The fields **Cumulated**, **Label**, **Path**, and **Unit** are non-editable and display the preconfigured values for the selected counter.

Result

In the **Preview** section, the values in the graph or table change as and when you change the filter options.

8. Click **Apply** to apply the changes that you made for the filters.
9. Click **Save** from the menu to save the data that you filtered.
10. Click **Edit** from the menu to exit the edit mode.

Results

You have filtered the data on the specific page for the report.

Customizing the appearance of graphs in a report

To display the data in a table, bar chart, or line chart in a manner that caters to your test requirements, use the controls that are available in the View Options of a report.

1. In the Test Navigator, expand the project until you locate the run.
   Each run begins with the name of the schedule or test, and ends with the date of the run in brackets.
2. Double-click the run.
   Result
   The default report opens.

3. Click the **Menu** icon 

and click the **Edit** icon.
4. Click the **Settings** icon  for the graph or table to modify.
5. The controls that are available in the View Options section depend on the graph type: bar chart, line chart, or table. For each graph type, only the applicable controls are displayed. You can adjust the following controls:

Option	Description
Adapt Y Scale	To compute minimum and maximum limit on the Y axis, select the check box. (all charts)
Title	Specify a title to the graph.
Show title	To hide the title, clear the check box.
X Axis Main items	Select the item to view on the X Axis.
Stacked Items	Select the item such as Pages or Time Ranges to view them in stack instead of separate bars.
Adapt Y scale on zoomed data	To adjust the Y scale according to zoomed data, select the check box. (line charts)
Show time ranges	To display the time range in the background of the chart, select the check box.
Line smoothing	To apply corners, clear the check box.
Orientation	To view bar charts horizontally or vertically, select an orientation.
Labels display policy	To hide the labels in a bar chart, select **Hidden**. To be able to accommodate labels within the frame of a bar chart, select **Adaptative**. If you select **Fixed**, long labels might not be visible.
Time line visibility	To view the time line of the chart in partial or full view, select **Small** or **Full** options. Drag the time line to create a new time range. If those options are specified, you can drag and create a new time range on the chart itself. If you select **None**, the time line is not visible and you cannot create a new time range on the chart.

6. After making the changes, click **Apply** and from the Menu click **Save**.
   To apply the changes to other reports, you can export the report definition and import it back. See Exporting report metadata on page 1490.

Changing the report displayed during a run

Use this page to select the default report that opens during a run. Typically, you select **Determine default report based on protocols in test**, which determines the protocols that you are testing and automatically opens the appropriate protocol-specific reports.

1. Open the Default Report Preferences page. Click **Window > Preferences > Test > Performance Test Reports > Default Report**.
2. In the Default Report window, select **Determine default report based on protocols in test** or a specific default report to display a customized report or if the default reports do not meet your needs. Note, however, that you will have to change this setting when you record other protocols.
3. Click **Apply**, and then click **OK**.

Modifying counters in a graph

To gather additional information for diagnosing performance problems, you can modify the counters that are displayed in a graph.

About this task

Counters are specific in-built queries that gather statistical information from the recorded test. The information can be the number of page hits, response time, and user load. By default, each report has pre-defined counters. You can add or remove the counters from the graphs in the report.

1. Double-click the report from the **Test Navigator** to modify the counters.
2. Click the **Menu** icon 

, and then click **Edit**.
3. Click the **Settings** icon  to modify counters on a specific graph.
4. Select the **Counters** tab on the **View Settings** page, and then perform the following steps to add, remove, or move the counters in a graph:
    a. Click the **Plus** button  , and then select the counters from the drop-down list to add a counter.
    b. Click the **Remove** button  to remove the selected counter.
    c. Use the **up-down** control buttons to move a counter.

    The **Preview** section displays the result of the actions.
5. **Optional:** For a selected counter, you can change the component of the counter. Based on the counter selection, the **Component** field shows the options available for that counter.
6. Perform the following steps to define a percentile value as decimal number for the counter:
    a. Select the **Percentile** as component from the **Component** drop-down list.
    b. Enter a new value in the **Percentile value** field.
    For example, *99.9*.

7. **Optional:** You can change the **Cumulated** value for the selected counter if you want to show the cumulation values on a graph. Select one of the following options based on the requirement:
   Choose from:
   - Select **No** to display the value of the last interval on the current time range.
   - Select **From the beginning of the time range** to display the cumulation of all values of the current time range.
   - Select **From the beginning of the run** to display the cumulation of all values from the beginning of the run to the end of the current time range.

   Notes:
   - For line charts, the default value is **No**.
   - For bar chart, pie chart, and tables, the default value is **From beginning of time range**.
   - The fields **Label**, **Path**, and **Unit** are non-editable.

8. Click **Apply**.
9. Click **Save** from the menu.
10. **Optional:** Click **Save As** to create another report with these changes.
11. Click the **Edit** icon to exit the edit mode.

Results
You have updated the counter information for the specific report.

Correcting time offset

Response time breakdown and resource monitoring data is time stamped using the system clock of the host computer. If there are differences between the system clocks of the host computers that you include in a test, then response time breakdown and resource monitoring data are skewed in reports. The best practice is to synchronize the system clocks on all computers that you include in a test. When this is not possible, you can correct the time offset of each host computer after a test run. Typically, correct the time offset on all computers to match the system clock of the workbench computer.

After you run tests with resource monitoring or response time breakdown enabled, follow these steps to correct the time offset:

1. In the **Test Runs** view, right-click the host where you want to correct the time offset; then click **Correct Time Offset**.
2. Select a **Shift Direction** of positive or negative. A positive shift moves the response time breakdown and resource monitoring data on the selected host to the right. A negative shift moves the response time breakdown and resource monitoring data on the selected host to the left.
3. Type the hours, minutes, or seconds of the time offset you want to use, and click **OK**.

Results

The response time breakdown and resource monitoring data on the selected host displays with a corrected time offset.

Export test results

You can export the test result in different formats to share it with different stakeholders.

Creating an executive summary from the workbench

To create a printable report that summarizes the findings of the performance test run on a single view, create an executive summary. You can export the data of the test run as an executive summary from a single report or from multiple reports such as Performance Report, Mobile and Web UI Statistical Report, Transaction Report, and Loop Report. You can then open the summary in a word-processing program to further format and annotate the data.

About this task

You export the executive summary to a local or a shared directory. You can export a test run from the Web Analytics report, from the test workbench, and from the command line.

When you use the workbench approach to create an executive summary, you can choose to create the summary for multiple runs and multiple report types at the same time. When you use the Web Analytics reports or the command line, you create executive summary for a particular run and a report at a time.

To create an executive summary from the workbench:

1. Click **File > Export > Test > Executive Summary**. You can also right-click the runs to create executive summaries for from the Test Navigator view and click **Export > Test > Executive Summary**. Each run would have one executive summary.
2. In **Export Directory**, specify the folder path to save the executive summary and click **Next**.
3. Select the runs to create the executive summary for. To create an executive summary for comparing two runs, select the **Generate a compare report** check box and select the main run to compare the report with and click **Next**.
4. Select a report to export and click **Finish**.

What to do next

A folder with the name of the run is created on the specified folder. To view the executive summary, open the `index.html` file.

Creating an executive summary from the Web Analytics report

To create a printable report that summarizes the findings of the performance test run on a single view, create an executive summary. You can choose to view the executive summary on a web browser or save it on a computer.

About this task

To generate an executive summary for a particular report such as Transaction report or Performance report, open that report and then follow the steps in this topic. To generate an executive summary for multiple reports or test runs at the same time, see Creating Executive Summary from Workbench on page 1487.

To create an executive summary from the Web Analytics report:

1. Open the test run to create executive summary for. The test run opens in a web browser.
2. From the dropdown, open the report for which to create executive summary.
3. Click the **Menu** icon 

, click the **Share** icon  , and click **Executive Summary**.
4. To view the executive summary of the report in another browser tab, click **View on another tab or page of the browser**. To save the executive summary, click **Save as an HTML file on the local computer**.
5. Click **Generate**.

Exporting reports to HTML format

When you export a test run and share it, people can analyze test data without using the test workbench. You can also email the test run or post it on a web server. The exported run can be displayed and printed from any browser. A test run contains multiple reports. You can choose to export any or all of the reports.

About this task

You can export a single run to a local directory or multiple runs in the compare mode to a directory. In addition to exporting a test run from Web Analytics, you can export it from the test workbench itself and from command line.

To export from the workbench, select a single run or multiple runs and click **Export > Test > Performance Test Run Statistics as HTML application** . To generate a single report comparing multiple runs, in the Export wizard, select the **Generate a compare report** check box and select a base run from the dropdown. To generate one report for each run, do not select the check box.

To export from Web Analytics:

1. Open the test run to export.
   The test run opens in an external or internal web browser.
2. Click the **Menu** icon 

, click the **Share** icon  , and click **Export Session to HTML**.
3. Select the type of report to export and click **Export**.
4. When you export from the workbench, specify a path to the folder to save the exported report.
   Your current project is the default save location. You can create a folder outside of the project to store exported reports.

   When you export from an external browser, the report is compressed and saved to the default download location of the browser.

What to do next

You can now share the test run with others. You can also export the test run from command line.

Related information

Configuration of test runs from the command line on page 1381

Exporting results to a CSV file

To further analyze test results, you can export all statistics or specific statistics captured during a run to a CSV file.

About this task

You can export a single run to a local directory or multiple runs in the compare mode to a directory. You can export the runs from Web Analytics report, workbench, and command line. To export from the workbench, select a single run or multiple runs and click **Export > Test > Performance Test Run Statistics as CSV File**. To export data of specific time ranges, on a subsequent page select a time range.

To export the run from command line, see the parameters in the Configuration of test runs from the command line on page 1381 topic.

1. Open the test run to export.
2. Click the **Menu** icon 

, click the **Share** icon  , and click **Export Session to CSV**.
3. Select the encoding system for the export.
4. Complete either one of the following steps:

   Choose from:

   - To export only the last value of each counter from the results or to export data of specific time ranges, select **Simple**.

     Note: When you export data of specific time ranges, for example, 5 Users or 15 Users, a separate column is created in the CSV file for each time range.

     - To create multiple CSV files if the number of columns exceed the specified value, select the **Split output if column exceeds** check box and specify a value.
   - To export all of the data for the run, select **Full**.

     To include description about the name of the run, node name, and time range for each counter, select the **Include per instance counters**.

- To export data of each location (agent) in a separate section in the CSV file, select the **Export each agent separately**

  To export data of each location (agent) to separate CSV files, select the **One file per agent** check box.

5. Click **Export**. If you export from the workbench, the report is saved in the specified folder. If you export from an external browser, the report is downloaded in a compressed format to the default download location of the browser.

What to do next

You can now analyze and share the report with people who are not using the workbench.

Related information

Exporting reports to HTML format on page 1488

Sharing URL of test run

When you share the URL of the test run with other people, they can view and analyze the test results on a browser on their computer if the test workbench is running on your computer at that time.

To share the URL of the test run:

1. Open the test run to share.
2. Click the **Menu** icon 

and click the **Share** icon  and select **Share Execution Result URL**.
   A unique URL is created for the test run.
3. Copy the URL and click **Close**.

What to do next

You can now share the URL of the test run with anybody.

Exporting report metadata

To share report metadata with another test workbench user, export the report definition. Use this option to share customized report formats with other users. The recipient imports the metadata with Eclipse's **Import** option and views the report from the Test Navigator or in the list of reports in the web report.

To export report metadata:

1. Click **File > Export**.
2. In the **Export** window, expand the **Test** folder, select **Report Definitions**, and click **Next**.
3. In **Save to File**, select the file that will contain the report. This file is created if it does not exist.
4. In **Select Report**, select the report to export, and then click **Finish**.
   The file is saved in the `.report` format.

What to do next

To apply another report definition to your reports, import that report metadata by clicking **File > Import > Report Definition**, and browse to the `.report` file.

Evaluating mobile test run results

To check whether or not the mobile test ran successfully, you can open the test report. You can also view each recorded functional action in the report.

About this task

When you run a test from Rational® Functional TesterHCL OneTest™ UI, you can view both the mobile web report and the statistical report. By default, the mobile web report is displayed after the run. You can also view this report on the mobile device.

To open the mobile web report and statistical report, from the Test Navigator view double-click a result from the Results folder.

When you run a test from the mobile device or emulator, at the end of play back, the report opens up automatically on the device or emulator. After the run, the report is uploaded to Rational® Functional TesterHCL OneTest™ UI automatically. There is no statistical report for the test that is run from the device or emulator.

The report is in a tabular format and displays the application that was tested, its execution status, and duration of the test and the measured response time. Each action is displayed in a row with the screen capture of the action highlighted and the time taken for that action from the beginning of the test.

If you added verification points to the test, you can also view the verification points entries in the report. The Execution Status of the report displays Failure, if the verification points fail.

Evaluating results in the web browser

When a test is run from IBM® Rational® Quality Manager, the result is displayed in a web browser. You can also change the default behavior of test workbench to view the results in a web browser within the workbench.

Customizing reports

To view reports in the manner that you want, you can customize a report. For information about the options available to customize a report, see Creating an executive summary from the Web Analytics report on page 1487.

Exporting reports

To create executive summary and export reports in HTML and CVS format, see Export test results on page 1487.

Logs overview

Rational® Functional TesterHCL OneTest™ UI uses logs to store different types of information, which you can use to determine the reason for a test failure.

Rational® Functional TesterHCL OneTest™ UI has the following logs:

Test logs

The test log contains a historical record of events that occurred during a test run or a schedule run, as well as the status of each verification point. The test log sets a verdict for each run as follows:

- **Pass** indicates that all verification points matched or received the expected response and all the test steps successfully completed. A verification point is set to PASS when the recorded/expected property value is received during playback. A Web UI test step is set to PASS when it has been executed successfully on the specified UI object.
- **Fail** indicates that at least one verification point did not match the expected response or that the expected response was not received or a Web UI step did not run successfully.
- **Error** indicates one of the following results: a primary request was not successfully sent to the server, no response was received from the server for a primary request, or the primary request response was incomplete or could not be parsed.
- The verdict is set to **Inconclusive** only if you provide custom code that defines a verdict of **Inconclusive**.

The verdict is rolled up from the child elements to the test level. For example, if a user group contains 25 virtual users, and five virtual users have failed verdicts, that user group has only one failed verdict, not five.

The test log file is stored in binary format with a `.executiondlr` file name extension in the project directory of your workspace. You can also view the test log in the user interface.

For more information about viewing test logs, see Viewing test logs on page 1493.

Problem determination logs

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 the Support person.

For more information about setting problem determination level, see Setting the problem determination level on page 1496.

Agent logs

Look in `%TEMP%` directory for the majordomo.log file. This file contains information about the attempts to contact the workbench including information about any failures and the reason for the failures.

On the Microsoft™ Windows operating system, the `%TEMP%` directory is typically at `%USERPROFILE%\AppData\Local\Temp`.

If the majordomo service is configured to log in as Local System Account, then the `%TEMP%` directory is at `%SystemRoot%\TEMP`, typically `C:\Windows\TEMP`.

Test log overview

The test log contains a historical record of events that occurred during a test run or a schedule run, as well as the status of each verification point. The test log for SAP tests contains the verdict of all the test steps in addition to the verification points.

About this task

The test log sets a verdict for each run as follows:

- **Pass** indicates that all verification points matched or received the expected response and all the test steps successfully completed. A verification point is set to PASS when the recorded/expected property value is received during playback. A Web UI test step is set to PASS when it has been executed successfully on the specified UI object.
- **Fail** indicates that at least one verification point did not match the expected response or that the expected response was not received or a Web UI step did not run successfully.
- **Error** indicates one of the following results: a primary request was not successfully sent to the server, no response was received from the server for a primary request, or the primary request response was incomplete or could not be parsed.
- The verdict is set to **Inconclusive** only if you provide custom code that defines a verdict of **Inconclusive**.

The verdict is rolled up from the child elements to the test level. For example, if a user group contains 25 virtual users, and five virtual users have failed verdicts, that user group has only one failed verdict, not five.

Viewing test logs

To see a record of all the events that occurred during a test run or a schedule run, as well as the status of each verification point, open the test log for that run. You can also compare an event from the test log with the request or response in the test to view the differences between the recording and the playback of the test.

About this task

The test log file is stored in binary format with a `.executiondlr` file name extension in the project directory of your workspace. You can also view the test log in the user interface.

1. In the **Test Navigator** pane, expand the **Results** folder to locate the test execution result.
2. Right-click the executed test result name, and then click **Display Test Log**.
3. On the **Overview** tab, view the verdict summary for the executed test. To see the potential data correlation errors in a separate view, click **Display Potential Data Correlation Errors**.
4. On the **Events** tab, view the errors, failures, and passes for each event in the test.
    - To navigate to the verdict type, click the **Select the verdict type**  icon.
    - To compare an event or request in the test log with the response or request of the test, right-click an event and click **Compare With Test > Request**, **Compare With Test > Response Data**, or **Compare With Test > Response Headers**. The compare editor highlights the differences.

5. On the **Data Correlation** tab, see all the references and substitutions that occurred during a test execution, as well as the data correlation errors. By default, you view both references and substituters. To view only substituters, click the **Show References** icon. To view the correlation data for each virtual user that was executed, click the **Merge Users** icon. This icon is enabled only for a schedule. In the **Data Correlation** section, when you click an event, you can see the correlation data in either the Content View or the Table View.

What to do next

From the test log, you can submit, search, and open defects in a defect tracking system. For details on configuring the test log preferences and working with defects, see Associating defects with a test log.

Viewing errors while running tests

To view errors and other events while a test is running, use the Execution Event Console view. If problems occur in a test run, you can examine the Execution Event Console view to determine whether to stop or continue the test.

1. Open the **Execution Event Console** view by clicking **Window > Show View > Execution Event Console**.
2. In the **Execution Event Console** view, click the **Filters** toolbar button in the upper, right corner.
   Result
   The **Event Console Configuration** window opens.
3. Select the types of messages and verdicts to display in the event console, and then click **OK**.
   You can also limit the number of events that are displayed per user and per run, and you can limit events to specific user groups or agent computers (locations). To configure other settings for the event console, click **Settings**.
4. Run performance tests as you normally do.
5. While a test is running, double-click an event in the **Execution Event Console** view to open the **Event Details** window.
   a. To change the order in which events are listed, click the **View Menu** toolbar button, and then select **Group By**.
6. To load events from the test log, ensure that the Test Log view is open and in the Console view, click the **Load Test Log Events** icon.

Exporting test logs

To process data from a performance test in another application or to use search tools to locate text in a test log, export the test log to a text file.

1. In the Test Navigator, right-click the run, and select **Export Test Log**.
   a. **Optional:** To export only a portion of the test log, open the test log by right-clicking the test run and then selecting **Display Test Log**. Right-click the elements to export, and then select **Export Log Element**.
   Result
   The **Export Test Log** window opens.
2. In the Export Test Log window, specify a location for saving the file, and then select options as follows:

Option	Description
Export format	Select default encoding or Unicode encoding.
Include event time stamps	Select to include event time stamps.
Include detailed protocol data	Select to include detailed protocol data. This option is available only for HTTP test runs.
Include response content	Select to include response content. This option is available only for HTTP test runs.
Include known binary data	Select to export binary data. This option is available only for HTTP test runs.

3. Click **Finish**.

   Result

   The test log is exported to a text file.

Exporting event log

To view all the events that occurred during the run of a test from another file, you can export this data from the Event Log panel, to an XML, CSV, or text file.

Before you begin

You must run a test to view data in the Event Log panel.

1. On the Event Log panel toolbar click the **View Menu** arrow icon ▽ and select **Export Event Log**.
2. In the Save dialog box, specify the location and format in which you want to save the events.

Exporting event console output

To view errors and other events of a test run from another file, you can export this data from the Execution Event Console view to an XML, CSV, or text file.

Before you begin

- Ensure that the Execution Event Console view is open by clicking **Window > Show View > Execution Event Console**.
- Ensure that the test is run and the Execution Event Console view contains data.

1. From the Execution Event Console view toolbar, click the **View Menu** arrow icon ▽ and select **Export**.
2. In the Save dialog box, specify the location and format in which you want to save the events.

Setting problem determination level for tests

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 the Support person.

About this task

The problem determination logs contain internal information about the playback engine. These logs are particularly useful for debugging problems such as Kerberos authentication, SSL negotiation, and resource constraints on an agent. The log files are named `CommonBaseEvents00.log` and are located in the deployment directory. For example, if you play back a schedule on an agent and set `C:\Agent` as the deployment directory, the problem determination log files are in a directory similar to `C:\Agent\deployment_root\<UserName>\A1E14699848784C00D2DEB73763646462\CommonBaseEvents00.log`. If a large amount of log information is generated, multiple `CommonBaseEvents` files are created.

1. Open the test for which you want to set the problem determination log level.
2. Select the root node and from the **Test Details** section, select **Problem Determination**.
3. On the Problem Determination page, set **Problem determination log level** to one of the following options:

All, Finest, Finer, Fine	Set these options only if you are requested to do so by technical support.
Config	Logs static configuration messages. Configuration messages, which include hardware specifications or system profiles, require no corrective action.
Info	Logs informational messages. Informational messages, which include system state, require no corrective action.
Warning	Logs warning messages. This is the default setting. Warning messages, which might indicate potential problems, require no corrective action.
Severe	Logs critical and unrecoverable errors. Critical and unrecoverable messages interrupt normal program execution, and require corrective action.

None	Turns logging off.

4. Save the test.

Results for tests in Functional Test perspective

In this section, you will learn how to analyze test results in the Functional Test perspective.

Functional test logs

After the playback is complete, you can view the results in the log. The results include any logged events such as verification point failures, script exceptions, object recognition warnings, and any additional playback information.

You can view Functional Test logs by setting the preferences in Logging page on page 834 .

Types of logs

You can use different types of Functional Test logs to view your playback results. These logs contain the same information in different formats. For more information, see Logging page on page 834.

Location of logs

When you set the log type to **HTML** or **Text**, Rational® Functional TesterHCL OneTest™ UI stores these logs in a log folder in the same location as the Functional Test project, but not in the Functional Test project. The name of the log folder is the project name with a suffix of **_logs**. For example, if your Functional Test project is **CalendarApp**, Rational® Functional TesterHCL OneTest™ UI stores its HTML or text logs in the **CalendarApp_logs** directory. You can open these logs from Rational® Functional TesterHCL OneTest™ UI in the Projects view. If you select **HTML**, your default browser opens the HTML log file. If you select **Text**, the text log file opens in the Functional Test script window.

In the Projects view, the HTML and text logs are listed within each project. Each script in the project has its own node in the logs directory. You can right-click the script log node to open, rename, delete, import, export, or view any logs or verification points.

To set the product to open the HTML and XML logs in a web browser, use the following commands:

- Internet Explorer :

```
assoc.html=htmlfile
```

- Mozilla Firefox:

```
assoc.html=FirefoxHTML-308046B0AF4A39CB
```

- Google Chrome:

```
assoc .html=ChromeHTML
```

Managing logs

You can use the Projects view to manage HTML or text logs. The logs appear below the project directory. Each script in the project has its own node under the logs directory. You can select a log and right-click **Open Log**, **Final Screen Snapshot**, **Delete**, or **Rename**. If your script has a verification point, you can select the log and open it in the Verification Point Editor by clicking **Open VP**, or open it in the Verification Point Comparator by clicking **Open Comparator**.

You can view the results of a verification point from the HTML log. At the end of each verification point entry in the HTML log is a **View Results** link. You should click this to open that verification point in the Verification Point Comparator. If the verification point fails, the baseline and actual files are shown beside one another which allows you to compare the data. If you receive an error about the Java™ plug-in while trying to start the Comparator, you must verify whether your plug-in is configured properly. For instructions, see Enabling the Java Plug-in of a Browser on page 800.

Note: To view the Rational TestManager version that can be integrated with Rational® Functional TesterHCL OneTest™ UI, see List of supported domains for functional testing by releases of Rational® Functional TesterHCL OneTest™ UI tech. note.

Logging page

You use the Logging page to set log and comparator options, such as preventing the script launch wizard from displaying on playback, displaying the log viewer after playback, and displaying a message before overwriting an existing log. You also use this page to indicate the type of log generated.

To access the logging page, click **Window > Preferences**. In the left pane, expand **Functional Test > Playback** and click **Logging**.

Note: For Microsoft Visual Studio, click **Tools > Options**. In the left pane, expand **Functional Test > Playback** and click **Logging**.

The Logging page contains the following options:

Don't show script launch wizard: When selected, prevents the script launch wizard from displaying each time you play back a script.

Display log viewer after script playback: When selected, this option displays the log after you play back a script. If the log type is HTML, the log opens in your default browser. If the log type is Text, the log opens in the Script Window of Rational® Functional TesterHCL OneTest™ UI. If the log type is XML, the log opens in your default browser.

Generally, the log file opens in the default browser that is associated with the html file extension in your computer. To view the html files in your desired browser, you can associate the html file extension with the specific browser. The file extension for different browsers are as follows:

- For Google Chrome, you must associate .html=ChromeHTML
- For Internet Explorer, you must associate .html=htmlfile
- For Firefox, you must associate .html=FirefoxHTML-308046B0AF4A39CB

Log screen snapshot for each action on the application: When selected, this option records a screen snapshot in the playback log against every action performed on the application.

Prompt before overwriting an existing log: When selected, this option prompts you before you overwrite a log.

Log the count of test objects created/unregistered at particular script line: When selected, this option logs these details:

- Number of objects created and unregistered at a specific script line
- Total number of objects created and unregistered per call script
- A cumulative number of test objects created and unregistered for the whole script during playback if Rational® Functional TesterHCL OneTest™ UI scripting methods have been used to return test objects.

Warning messages are also logged at the call script level and the main script level, if the number of test objects created exceeds the number of test objects unregistered, which would suggest the possibility of memory leaks during playback.

Log a screen snapshot when playback fails: When you select this option, it captures a screen snapshot at the time of the failure and stores it in the log. You must clear the check box to save storage space (172 KB per snapshot).

Log GUI actions on the application: When you select this option, it adds a detailed record of any GUI-related actions performed on the application (without a screen snapshot) to the playback log.

Log type: This option Indicates the type of log Rational® Functional TesterHCL OneTest™ UI generates to write results of script playback. The log types are as follows:

- **None**: Generates no log, if selected.
- **Text**: Displays the log in ASCII format in the Functional Test script window.
- **HTML**: Displays the log in HTML format in your default browser. The left pane in the HTML log contains three boxes: Failures, Warnings, and Verification Points. The list of items in each box help you navigate to a specific location in the log. You can select an item to quickly find important errors, warnings, and verification point results in the log. To do so, double-click an item in a list, and Rational® Functional TesterHCL OneTest™ UI scrolls to and displays the item in the log.
- **TPTP**: Displays a log using TPTP in the Functional Test script window.
- **XML**: Displays a log of XML data rendered in HTML format [using transformation and Cascaded Style Sheets] in your default browser.
- **Default**: Displays the unified report for the test scripts in the browser window. This is also the default option to generate result for Functional test scripts.

- **JSON**: Displays a log in JSON format in the Functional Test Script window. Each event in this log type is a separate JSON.

  Note: The JSON log type is not supported in the integration of Rational® Functional TesterHCL OneTest™ UI with Visual Studio.

Use Default: Clear the check box to change the value in the **Log type** field. Select the check box to restore the default value.

Restore Defaults: Restores the default values on this page.

Apply: Saves your changes without closing the dialog box.

Setting log preferences

You can set your log preferences to view functional test logs.

1. Click **Window > Preferences**.Click **Tools > Options**.
2. On the left pane, expand **Functional Test** , expand **Playback**, and click **Logging**.
3. Select **Display log viewer after script playback** to open the log automatically after playback.
4. Select other logging options as required.
5. Set the viewing preferences in the **Log type** list:

   a. By default, the log type is **XML**. To change the log type, clear the **Use Default** check box, and select **Text**, **HTML**, **XML**, or **JSON**.
      - If you select **Text** or **JSON** , the text log file opens in the Functional Test Script window.
      - If you select **HTML** or **XML**, the HTML or XML log file opens in your default browser.

      Notes:
      - To view the results of a particular verification point in the HTML log, click **View Results** at the end of a verification point entry. The Verification Point Comparator displays the baseline and actual files side by side if the verification point failed, so that you can compare the data.
      - If you get an error about the Java plug-in when you click the **View Results** link in the HTML log, verify that your plug-in is configured correctly. If you select **XML**, the XML log file opens in your default browser.
      - Ensure that the functional testing product supports the default browser version.

   b. If you select **TPTP**, the TPTP log file opens in the Functional Test Script window.

Note: You can only view TPTP log files in the Rational® Functional TesterHCL OneTest™ UI Eclipse Integrated Development Environment, not in the Microsoft Visual Studio Integrated Development Environment. Customized or extended log files in other formats cannot be viewed in either the Eclipse IDE or the Visual Studio IDE.

c. Click **Apply** to save the new settings and to continue changing options or click **OK** to save the new setting and to close the Preferences dialog box.

Disabling enhanced log results

Logged events such as verification point failures, script exceptions, object recognition warnings, and other additional playback information are displayed in the playback log results. From Rational® Functional TesterHCL OneTest™ UI version 8.2 and later, the results of the getProperty() command are also displayed in the log results. If you do not require the log event to be displayed in the playback log, you can disable the event in the log results.

Before you begin

Ensure that you have access to modify the `ivory.properties` file.

About this task

To disable the getProperty() log event, you must modify the `ivory.properties` file.

1. Open the `ivory.properties` file available in the `<installation_folder>\bin\` directory.
2. Add the following line of code at the end of the file contents:`rational.test.ft.log.enhanced=false`

Note: To enable the getProperty() log event again, set `rational.test.ft.log.enhanced=true`.

Viewing logs in the Projects view

You can open a log from the Projects view. In Projects view, the HTML, XML, and text logs are listed within each project. The log list is displayed below the project. The log has the same name as the project, with **_logs** appended, for example, *projectname*_logs.

Do one of the following procedures:

- To view a log, select a log in the Projects view and right-click **Open Log**.
- To open the log in the Verification Point editor, select a log in the Projects view and right-click **Open VP**.
- To open a log in the Verification Point Comparator, select the log in the Projects view and right-click **Open Comparator**.

Note: You can only view TPTP log files in the Rational® Functional TesterHCL OneTest™ UI Eclipse Integrated Development Environment, not in the Microsoft Visual Studio Integrated Development Environment. Customized or extended log files in other formats cannot be viewed in either the Eclipse IDE or the Visual Studio IDE.

Viewing Dojo logs

Dojo logs are based on XML logs and display a graphical representation of the test results. You can use Dojo logs to select filters and view verification points, failed verdicts, warning verdicts, and the detailed information on each action that is recorded in the script. Dojo logs open in the default browser after script execution.

1. Start Rational® Functional TesterHCL OneTest™ UI.
2. Record a script.
3. Click **Window > Preferences**.
4. On the left pane, expand **Functional Test > Playback**.
5. Click **Logging**.
6. Ensure that the **Use Default** check box is cleared.
7. Select **xml** from the **Log type** list, and click **OK**.
8. Playback your script. The Dojo log is displayed in the default web browser.

Note: For detailed information about *viewing Dojo logs in the Firefox browser*, see Unable to view Dojo logs in Firefox version 3.0.

Renaming and deleting logs

You can rename and delete logs from the Projects view.

- To rename a log, select a log in the Projects view and right-click **Rename**.
- To delete a log, select a log in the Projects view and right-click **Delete**.

Log Extension

You can create customized Rational® Functional TesterHCL OneTest™ UI logs in addition to the standard log formats: text, HTML, Test and Performance Tools Platform (TPTP), and XML. You can use the customized logs to view your playback results.

Extending a log

You can customize your log files by configuring the plug-ins to extend an extension point and writing a class that designs your log.

1. Create a new plug-in project.
   Result

By default, the Eclipse environment displays the project **Overview** after you create the project.

Note: If the project **Overview** is not displayed by default, right-click **MANIFEST.MF** under the `META-INF` folder, and click **Open With > Plug-in Manifest Editor**.

2. Click the **Dependencies** tab, and click **Add** under **Required Plug-ins**.
3. Select **com.ibm.rational.test.ft.playback** from the **Plug-in Selection** list, and click **OK**.
4. Save the plugin.xml file.
5. Click the **Extensions** tab, and click **Add**.
6. Select the **com.rational.test.ft.playback.logExtension** extension point from the list, and click **Finish**.
7. Type the extension ID and Name.
8. Right-click **com.ibm.rational.test.ft.playback.logExtension** in the left pane of Extensions window.
9. Click **New > Log**.

   Result

   The log file is displayed under the extension file.

   Note: Customized or extended log files in formats other than the standard formats cannot be viewed in either the Rational® Functional TesterHCL OneTest™ UI Eclipse Integrated Development Environment or the Visual Studio IDE.

10. In **Extension Element Details**, specify the properties of the log.
    a. Type a unique ID for the log in **LogID** field. Rational® Functional TesterHCL OneTest™ UI uses this LogID to list the log type on the Preference Page.
    b. Type a class name in **Class** field. This class name extends the base class LogExtensionAdapter.
    c. **Optional:** Type a description of the log in **Description** field.
11. Save the project.

Deploying the extended log file

You can deploy your customized log file by exporting the plug-in in a format suitable for deployment.

1. In the project folder in the left pane of the window, right-click **plug-in.xml**.
2. Click **Export**.
3. Select **Deployable plug-ins and fragments** in **Plug-in Development** folder.
4. Click **Next**.
5. Select the plug-in from the **Available Plug-ins and Fragments** list.
6. Set the destination directory to export the selected projects for deployment.

   The default destination directory on Windows is `C:\Program Files\IBM\SDP` `C:\Program Files\HCL\hclonetest` and on Linux is `/opt/IBM/SDP` `/opt/HCL/hclonetest`.

   If the Eclipse environment is extended during installation of Rational® Functional TesterHCL OneTest™ UI, the log plug-ins need to be exported to the extended Eclipse location.

7. Click **Finish**.
8. In the Save Resource dialog box, click **Yes**.

Example: Creating a text log

The following example shows how to create a text-log output from an empty test script.

Implement the subclass that inherits the base class `LogExtensionAdapter` to implement the following methods to get the log result that you want.

```
public void initLog()
public void writeLog(ILogMessage message)
public void closeLog()
```

Here is an example of text-log output from an empty test script.

```
July 23, 2007 8:30:12 PM IST :Script Name Script1.java Result :INFO Event SCRIPT START headlind Script
 start [Script1]
Property Name =line_number Property Value =1
Property Name =script_name Property Value =Script1
Property Name =script_id Property Value =Script1.java
```

```
July 23, 2007 8:30:12 PM IST :Script Name Script1.java Result :PASS Event SCRIPT END headlind Script end
 [Script1]
Property Name =line_number Property Value =-1
Property Name =script_name Property Value =Script1
Property Name =script_id Property Value =Script1.java
```

Exemple

The following example shows an implementation of a text-log to get the text-log output that is shown in the previous example:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Vector;

import com.rational.test.ft.services.ILogMessage;
import com.rational.test.ft.services.LogException;
import com.rational.test.ft.services.LogExtensionAdapter;
import com.rational.test.ft.services.LogMessageProperty;

public class ExampleLog extends LogExtensionAdapter {

private String logName=null;
private String logDirectory=null;
private PrintWriter out=null;

public ExampleLog(String logName) {
 super(logName);
 this.logName=logName;
 this.logDirectory=null;
}
```

```
public ExampleLog() {
 super();
 this.logName=null;
 this.logDirectory=null;
}

/**
* Initialize the stream to write the log.
*/
public void initLog() throws LogException {
 try{
  this.logName=getLogName();
  this.logDirectory=getLogDirectory();
  File logFile=new File(logDirectory,logName+".txt");
  FileOutputStream fos=new FileOutputStream(logFile);
  out=new PrintWriter(fos);
 }catch(IOException e)
 {

 }
}

/**
* Write log events/message to the file stream
*/
public void writeLog(ILogMessage message) {
 Vector properties=message.getProperties();
  String result=getResult(message);
  String event_type=getEventType(message);
  String headline=getHeadline(message);
  String timestamp=getTimestamp();
  String currentScriptName=getScriptName(message);
  out.println(timestamp + " :Script Name " + currentScriptName + " Result :" + result + " Event " +
 event_type + " headlind " + headline );
  for(int i=0,size=properties.size();i<size;i++)  {
  LogMessageProperty property =
   (LogMessageProperty) properties.elementAt( i );
  out.println("Property Name =" + property.getName().toString() + " Property Value ="
 +property.getValue().toString() );
  }
  out.println();
}

/**
* Close the stream to persist the logs.
*/
public void closeLog() {
 try{
   out.close();
 }catch(Exception e) {

 }
}

/**
* Return the result from the log message.
*/
private String getResult(ILogMessage message) {
```

```
  String result=null;
  switch (message.getResult())
  {
   case LOG_FAILURE : result="FAILURE";break;
   case LOG_PASS : result="PASS";break;
   case LOG_WARNING : result="WARNING";break;
   default: result= "INFO";
 }
 return result;
}

/**
* Return string representation of event from the ILogMessage.
*/
private String getEventType(ILogMessage message) {
  String eventType=null;
  switch(message.getEvent())
  {
   case EVENT_SCRIPT_START : eventType="SCRIPT START";break;
   case EVENT_SCRIPT_END : eventType="SCRIPT END";break;
   case EVENT_VP : eventType="VERIFCATION POINT";break;
   case EVENT_CALL_SCRIPT : eventType = "CALL_SCRIPT"; break;
   case EVENT_APPLICATION_START : eventType="APPLICATION START";break;
   case EVENT_APPLICATION_END : eventType="APPLICATION END";break;
   case EVENT_TIMER_START : eventType="TIMER START";break;
   case EVENT_TIMER_END : eventType= "TIMER END" ;break;
   case EVENT_CONFIGURATION : eventType="CONFIGURATION"; break;
   default : eventType="GENERAL";
 }
 return eventType;
}

/**
* Returns the headline from the ILogMessage.
*/
private String getHeadline(ILogMessage message) {
  return message.getHeadline();
}

/**
* Return the script name from the ILogMessage.
*/
private String getScriptName(ILogMessage message) {
  String scriptName=null;
  Vector properties=message.getProperties();
  for(int i=0,size=properties.size();i<size;i++)  {
   LogMessageProperty property =
   (LogMessageProperty) properties.elementAt( i );
   if(property.getName().equalsIgnoreCase(PROP_SCRIPT_ID))
   {
     scriptName=property.getValue().toString();
   }
  }
  return scriptName;
}
}
```

Chapter 10. Troubleshooting

This guide describes how to analyze and resolve some of the common problems that you might encounter while you work with Rational® Functional TesterHCL OneTest™ UI.

Troubleshooting in the UI Test perspective

You can find information about the issues or problems that you might encounter while working with Rational® Functional TesterHCL OneTest™ UI in the UI Test perspective. Details about issues, their causes and the resolutions that you can apply to fix the issues are described in this section.

Unable to play back Web UI tests when certain web applications are redirected to a different URL

When you record a Web UI test and enter the URL of the web application that you want to test, the application might be redirected to a different URL. When you play back this recorded Web UI test, the playback fails.

Cause

When you record a Web UI test by starting a web application that redirects to a different URL, the following changes occur:

- The Web UI recorder captures the redirected URL instead of the URL of the web application.
- The **URL** field in the **Application Details** pane of **Launch application** is updated with the redirected URL while recording.

When you play back the recorded Web UI test, the URL of the **Launch application** node differs from the URL that you use to start the web application during the recording.

Resolution

In the recorded Web UI test, you must change the redirected URL to the URL of the web application in the **Application Details pane** of **Launch application**.

Unable to open a browser for the Web UI test playback

When you play back a Web UI test, the test fails occasionally before Rational® Functional TesterHCL OneTest™ UI opens a browser.

Cause

The Web UI test playback fails if the IP address of the workbench URL, which is automatically detected by the playback process, is incorrect.

Resolution

1. Click the **Display workbench URL to connect or reconnect a device** icon  on the toolbar.

   The **Workbench URL** dialog is displayed.
2. Select an alternate URL from the options available in the **Workbench URL** drop-down list.
3. Click **Close**.
4. Play back the test.

The AUT displays the 403 forbidden error

When you record or play back a Web UI test, the application under test (AUT) fails to open and displays the 403 forbidden error.

Cause

The Web UI testing fails because access to the web page is forbidden in the browser instance that is opened for test run.

Resolution

1. Follow these steps to fix the 403 forbidden error while you record the test in a browser:
   a. Select the **Web UI recording > Launch browser with arguments** option in the **Recorder Settings** dialog to specify the argument to be used when Rational® Functional TesterHCL OneTest™ UI opens the browser.
   b. Enter the following argument in the text box:
   ```
   --disable-blink-features=AutomationControlled
   ```
2. Follow these steps to fix the 403 forbidden error while you play back the test in a browser:
   a. Select the **Launch browser with arguments** checkbox in the **Run Configuration** dialog to specify the argument to be used when Rational® Functional TesterHCL OneTest™ UI opens the browser.
   b. Enter the following argument in the text box:
   ```
   --disable-blink-features=AutomationControlled
   ```

Unable to verify content in a Windows test

When you play back a Windows test with a content verification point, the test fails to verify the text content.

Cause

The content verification point step in the Windows test fails if the expected value is either set incorrectly or the text field is empty.

Resolution

1. Open the Windows test that has a content verification point in the **UI Test** perspective.
2. Click the step with a content verification point.

   The **Check Action Details** pane is displayed.
3. Perform the following steps:
   a. Select a condition to be used to verify the content from the dropdown list that is next to the **Verify attribute** field.

      For example, select **equals** to verify whether the received value is equal to the expected value.

   b. Check and correct the expected value to be verified in the text field that is next to the **Verify attribute** field.
4. Click **File > Save**.
5. Play back the test.

Unable to run Web UI tests on a remote agent on Linux

When you run Web UI tests on a remote agent that runs on Linux, the browser fails to launch.

Cause

Browser launch is prevented when the remote agent runs as *root* or *sudo*.

Resolution

To resolve this issue, you must perform the following steps after you have installed the remote agent:

1. Stop the agent.
2. Start the agent as the logged in user by performing the following steps:
   a. Open a terminal on Linux.
   b. Enter the following commands:

      cd /opt/IBM/SDP/Majordomo//opt/HCL/HCLOneTest/Majordomo/

      sudo ./MDStop.sh

      ./MDStart.sh

Unable to run Web UI tests on an Android 13 device or emulator

When you run Web UI tests in Google Chrome versions that are earlier than 110 on an Android 13 device or emulator, the browser fails to open.

Cause

The browser fails to open due to an issue with the Google Chrome driver for versions that are earlier than 110.

Resolution

To resolve this issue, you must perform the following steps:

1. Upgrade the existing version of Google Chrome to 110 or later on the Android 13 device.
2. Run the Web UI tests on the device or emulator.

Related information

Running a Web UI test on page 1243

Unable to generate tests in a Linux or Mac operating system

When you install Rational® Functional TesterHCL OneTest™ UI on a Linux or Mac operating system and create a test, occasionally, the test generation fails with NullPointerException.

Cause

The test generation fails if there is no write permission to the `/var/tmp` folder before you create any test.

Resolution

1. Log in to the computer that runs a Linux or Mac operating system as a sudo user.
2. Run the following command to apply the write permission to the `/var/tmp` folder:

   sudo chmod -R 777 /var/tmp

3. Navigate to the `/var/tmp` folder, and then run the following command to check the permission that is set:

   ls -ltr

Rational® Functional TesterHCL OneTest™ UI error messages

This section provides information about error messages that you might encounter with Rational® Functional TesterHCL OneTest™ UI. This section lists the error messages by ID, explanation, system action and your response to correct the error message.

CRRTWF0001E

There was an error in asserting the Rational® Functional TesterHCL OneTest™ UI log type.

Explanation: There was an error in asserting the log type of Rational® Functional TesterHCL OneTest™ UI in the test navigator.

System action: None

User response: Refresh the Rational® Functional TesterHCL OneTest™ UI project in the Functional Test perspective.

CRRTWF0002E

There was an error in getting an image for the Rational® Functional TesterHCL OneTest™ UI type.

Explanation: There was an error in getting the images for Rational® Functional TesterHCL OneTest™ UI assets

System action: None

User response: Refresh the Rational® Functional TesterHCL OneTest™ UI project in the Functional Test perspective.

CRRTWF0003E

There was an error while verifying if Rational® Functional TesterHCL OneTest™ UI is installed.

Explanation: There was an error while verifying if Rational® Functional TesterHCL OneTest™ UI is installed.

System action: None

User response: Start __WUT_PRODUCT_NAME__ in administrator mode. Alternatively, reinstall Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__

CRRTWF0004I

IBM® Rational® Test WorkbenchHCL OneTest™ Studio sending command to open Log

Explanation: This is a trace statement for debugging while opening the test script log file.

System action: None

User response: This message is for information only.

CRRTWF0005I

After sending command to open Log

Explanation: This is a trace statement for debugging while opening the test script log file.

System action: None

User response: This message is for information only.

CRRTWF0006E

There was an error in opening the Rational® Functional TesterHCL OneTest™ UI script.

Explanation: The Rational® Functional TesterHCL OneTest™ UI script could not be opened from the __TW_SUITE_NAME__.

System action: None

User response: Refresh the Rational® Functional TesterHCL OneTest™ UI assets in the Functional Test perspective.

CRRTWF0007I

IBM® Rational® Test WorkbenchHCL OneTest™ Studio sending command to open Script

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0008I

After sending command to open Script

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0009I

Opening IBM® Rational® Functional TesterHCL OneTest™ UI Script in IBM® Rational® Test WorkbenchHCL OneTest™ Studio Perspective

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0010I

Opening IBM® Rational® Functional TesterHCL OneTest™ UI Script in IBM® Rational® Test WorkbenchHCL OneTest™ Studio Perspective

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0011E

There was an error in opening the Rational® Functional TesterHCL OneTest™ UI script in the __WUT_PRODUCT_NAME__ perspective.

Explanation: There was an error in opening the Java editor in __WUT_PRODUCT_NAME__.

System action: None

User response: Open the project in a new Eclipse workspace

CRRTWF0012I

About to execute IBM® Rational® Functional TesterHCL OneTest™ UI Script

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0013I

After the IBM® Rational® Functional TesterHCL OneTest™ UI Script Execution

Explanation: This is a trace statement for debugging while opening the test script.

System action: None

User response: This message is for information only.

CRRTWF0014E

There was an error in executing the Rational® Functional TesterHCL OneTest™ UI script.

Explanation: A Rational® Functional TesterHCL OneTest™ UI script could not be executed from __WUT_PRODUCT_NAME__

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__

CRRTWF0015E

Shell-sharing of Rational® Functional TesterHCL OneTest™ UI with __WUT_PRODUCT_NAME__ might not be configured.

Explanation: There was an error in running certain actions for Rational® Functional TesterHCL OneTest™ UI.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__.

CRRTWF0016E

No selection is available to complete the operation.

Explanation: There was an error in determining the file that was selected from the test navigator.

System action: None

User response: Refresh the Rational® Functional TesterHCL OneTest™ UI Assets in the Functional Test perspective. Alternatively, reinstall Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__.

CRRTWF0017E

An invalid extension was defined for Rational® Functional TesterHCL OneTest™ UI integration.

Explanation: There was an error in running certain actions for the Rational® Functional TesterHCL OneTest™ UI assets.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__.

CRRTWF0018E

There was an error in sending requests to Rational® Functional TesterHCL OneTest™ UI.

Explanation: There was an error in running certain actions for the Rational® Functional TesterHCL OneTest™ UI assets.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__.

CRRTWF0019E

There was an error in obtaining the Rational® Functional TesterHCL OneTest™ UI test path.

Explanation: An error occurred in determining the location of the Rational® Functional TesterHCL OneTest™ UI asset that was selected from the test navigator.

System action: None

User response: Refresh the Rational® Functional TesterHCL OneTest™ UI assets in the Functional Test perspective. Alternatively, reinstall Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with __WUT_PRODUCT_NAME__.

CRRTWF0020E

There was an error in generating code for Rational® Functional TesterHCL OneTest™ UI.

Explanation: An error has been detected while generating code for Rational® Functional TesterHCL OneTest™ UI.

System action: None

User response: Add the functional test again to a new compound test.

CRRTWF0101E

Exception in Setting Test Path of IBM® Rational® Functional TesterHCL OneTest™ UI Test

Explanation: The path of the Rational® Functional TesterHCL OneTest™ UI script could not be determined.

System action: None

User response: Contact IBM®HCL® Support with the error log.

CRRTWF0102E

Exception in refresh of Editor Tree

Explanation: The test navigator could not be refreshed.

System action: None

User response: Open the project in a new workspace.

CRRTWF0103E

Exception in opening the editor

Explanation: There was an error in opening the test editor

System action: None

User response: Open the project in a new workspace

CRRTWF0104E

Exception while renaming Functional Test Script Assets

Explanation: There was an error in refreshing the test navigator assets while the Rational® Functional TesterHCL OneTest™ UI scripts were modified.

System action: None

User response: Refresh the test assets.

CRRTWF0105E

Exception while setting the job schedule for Renaming Functional Test Script Assets

Explanation: There was an error in refreshing the test navigator assets while the Rational® Functional TesterHCL OneTest™ UI scripts were modified.

System action: None

User response: Refresh the test assets.

CRRTWF**0201**I

Rational® Functional TesterHCL OneTest™ UI is not installed.

Explanation: Rational® Functional TesterHCL OneTest™ UI must be installed before you can run a compound test that contains Rational® Functional TesterHCL OneTest™ UI scripts.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell shared mode with Rational® Test WorkbenchHCL OneTest™ Studio.

CRRTWF**0202**I

Inflating Project during execution

Explanation: This information is used for debugging while extracting the test project.

System action: None

User response: None

CRRTWF**0203**E

There was an error while executing a Rational® Functional TesterHCL OneTest™ UI script.

Explanation: The Rational® Functional TesterHCL OneTest™ UI script could not be executed.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with Rational® Functional TesterHCL OneTest™ UI.

CRRTWF**0204**E

Exception in parsing Functional Log

Explanation: An error occurred in processing the log while executing the Functional Test Script.

System action: None

User response: Consider contacting support with the error log

CRRTWF**0205**E

A class could not be loaded during execution

Explanation: An error occurred in executing the Rational® Functional TesterHCL OneTest™ UI script.

System action: None

User response: Reinstall Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with Rational® Functional TesterHCL OneTest™ UI

CRRTWF**0206**E

A directory could not be created during execution

Explanation: There was an error in adding snapshots to the Rational® Functional TesterHCL OneTest™ UI log file.

System action: None

User response: Run Rational® Functional TesterHCL OneTest™ UI in administrator mode.

CRRTWF**0301**E

The execution results could not be parsed.

Explanation: There was an error in parsing the results of the Rational® Functional TesterHCL OneTest™ UI script playback

System action: None

User response: Contact Support with the error log file.

CRRTWF**0302**E

The property value of a verification point could not be fetched.

Explanation: There was an error in parsing the log events of the Rational® Functional TesterHCL OneTest™ UI script playback.

System action: None

User response: Contact Support with the error log file.

CRRTWF**0303**E

A command to get the verification point could not be created.

Explanation: There was an error in opening the verification point of the Rational® Functional TesterHCL OneTest™ UI script.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in the shell-shared mode with Rational® Functional TesterHCL OneTest™ UI.

CRRTWF0304E

There was an error in getting a command parameter that is needed to open the verification point comparison editor.

Explanation: A property required to open the verification point of the Rational® Functional TesterHCL OneTest™ UI script is missing.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with Rational® Functional TesterHCL OneTest™ UI.

CRRTWF0305E

The verification point comparison editor could not be opened.

Explanation: There was an error in opening the verification point of the Rational® Functional TesterHCL OneTest™ UI script.

System action: None

User response: Install Rational® Functional TesterHCL OneTest™ UI in shell-shared mode with Rational® Functional TesterHCL OneTest™ UI.

CRRTWM0001E

Missing message for log entry '*{0}*' in class: *{1}*

Explanation: A text message is missing for a loggable key.

User response: Contact IBM®HCL® Software Support if you cannot resolve the issue.

CRRTWM0002E

Cannot get Log key '*{0}*': SecurityException raised.

Explanation: Java VM raise a security exception when trying to check a loggable message.

User response: Contact your support.

CRRTWM0003E

Cannot initialize Log key '*{0}*'

Explanation: The Java VM raised an exception during initialization of a log message.

User response: Contact IBM Software Support if you cannot resolve the issue.

CRRTWM0004E

IExtendedType '*{0}*' already defined, at extension point '*{1}*'

Explanation: An extension is registered twice with the same ID used for the same extension or two different extensions.

User response: Contact IBM®HCL® Software Support if you cannot resolve the issue.

CRRTWM0008W

Warning: field '*{0}*' is not defined in class: *{1}*

Explanation: A log message is not defined by the class.

User response: Contact IBM Software Support if you cannot resolve the issue.

CRRTWM0009W

Warning: Cannot check a message against log key mapping for '*{0}*' of class *{1}*, SecurityException raised.

Explanation: The Java VM raised an exception the checking the validity of a log message key using existing text.

User response: Contact IBM®HCL® Software Support if you cannot resolve the issue; however, note that this will not affect the functioning of the product.

CRRTWM0010E

Unexpected exception, please check Error Log view: *{0}*

Explanation: An unexpected exception occurs during processing

System action: Workbench may not work properly depending on exception raised.

User response: Try again, contact your support if problem persist.

CRRTWM0011W

Mismatch between number of formal bindings and number of icu bindings for key *{0}*

Explanation: The number of types given to format ICU numbers correctly doesn't match the number of formal parameters for this translated message

System action: None.

User response: Nothing to do, product will work properly except for the ICU formatting of some of the parameters of this translated message in the report

CRRTWM0012W

Cannot parse number from string '*{0}*', a parameter of the translated message *{1}* for icu bindings

Explanation: This string is not a valid number

System action: None.

User response: Nothing to do, product will work properly except for the ICU formatting of some of the parameters of this translated message in the report

CRRTWM1001E

unexpected exception

Explanation: An exception that could not be handled occurs during processing.

System action: None.

User response: Close test editor and report exception to product support.

CRRTWM1002E

Error getting persistent property '*{0}*'

Explanation: Persistent properties are used to store some editor configuration and cannot be loaded.

System action: Editor may not restore some of editor configuration, but should works correctly.

User response: Nothing to do as editor should works correctly.

CRRTWM1003E

Error setting persistent property '*{0}*'

Explanation: Persistent properties are used to store some editor configuration and cannot be saved.

System action: None.

User response: Nothing to do as editor should works correctly.

CRRTWM1004E

Cannot reload device list

Explanation: Device editor is not able to reload device list.

System action: Device editor display empty device list.

User response: Close and restart workbench, open Device editor again, if problem persist, please contact your support.

CRRTWM1005E

Cannot reload application list

Explanation: Cannot reload application list

System action: Application editor display empty application list

User response: Close and restart workbench, open Application editor again, if problem persist, please contact your support.

CRRTWM1006E

Cannot save editor '*{0}*'

Explanation: For some reason an editor cannot be saved.

System action: Editor data are not saved, modified data will be lost.

User response: Close workbench and retry, if problem persist, please contact your support

CRRTWM1007E

Cannot reload resource '*{0}*'

Explanation: For some reason a workbench resource cannot be reloaded.

System action: None.

User response: Close editor and open it again, if problem persist, please contact your support.

CRRTWM1008E

IExtendedTypeUI '*{0}*' already defined, at extension id='*{1}*'

Explanation: An editor extension is registered twice, this must be fixed.

System action: Editor may not be able to edit right type of data.

User response: Report the error to your product support.

CRRTWM1009E

Failed to generate QRCode image, content is:'*{0}*'

Explanation: QRCode image cannot be generated.

System action: Workbench is not able to display QRCode image.

User response: Report the error to your product support.

CRRTWM1010I

Imported package '*{0}*' is not rebuild because application '*{1}*' already exists (*{2}*)

Explanation: The imported application already exist in your workbench and cannot be built.

System action: Application Editor display both (same) application.

User response: Open Application Editor to checks for both application, remove the already existing application. Before importing again, do not forget to remove imported application.

CRRTWM1011E

Cannot update original package of Web application: '*{0}*' (build dir: '*{1}*')

Explanation: User edits Web properties and Application Editor cannot save changes in original package nor update managed application workspace resource. Note that Application Editor will display right properties values.

System action: Application Editor display right properties values, but user won't be able to rebuild application with that properties in the future.

User response: Check if build exists and is accessible.

CRRTWM1012E

Mobile Run Wizard extension already defined for '*{0}*'

Explanation: More than one feature provide Mobile Run Wizard extension with same id.

System action: Extension is ignored, you may missing feature when using Mobile Run Wizard

User response: Report the error to your product support.

CRRTWM1013E

Unable to run external command '*{0}*'

Explanation: Product failed to try to run an external command.

System action: Subsequent action are ignored.

User response: Report the error to your product support.

CRRTWM1014E

Unable to handle menu from report: reason '*{0}*'

Explanation: Product failed to handle a menu from report.

System action: None.

User response: Depending on the given reason, you may be missing the test suite corresponding to this UI report, or it may have been moved to another location in the workspace.

CRRTWM1015E

Failed to download the driver for '*{0}*' browser.

Explanation: Product failed to download compatible driver version for browser.

System action: None.

User response: Report the error to your product support.

CRRTWW0031E

Exception in starting Edge browser.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWM1101E

unexpected exception

Explanation: An unexpected exception occurs during processing.

User response: Try again, and contact your product support if issue continue.

CRRTWM1102E

IExtendedTypeTegGen '*{0}*' already defined, at extension id='*{1}*'

Explanation: A test generation extension is registered twice.

User response: report the issue to your product support.

CRRTWM1201E

Execution exception

Explanation: An exception during the execution

User response: contact your support.

CRRTWM1202E

Unexpected exception

Explanation: An unexpected exception occurs during processing

User response: Contact your support.

CRRTWM1203I

Information: %1

Explanation: This message display an information from execution engine.

User response: Check information message.

CRRTWM1206E

Exception thrown during server address check

Explanation: An exception is raised when checking the address of a server

User response: Contact your support

CRRTWM1210E

Exception thrown during data harvest

Explanation: An exception is raised during the processing of a data harvest

User response: Check exception message, try again or contact your support.

CRRTWM1211E

Exception thrown during substitution

Explanation: An exception is raised during the processing of a substitution

User response: Check exception message, try again or contact your support.

CRRTWM1301E

Unexpected exception

Explanation: An unexpected error occured during processing.

System action: None.

User response: Contact __VENDOR_NAME__ Support.

CRRTWM1302I

Android USB controller process exited with exit code *{0}*\n*{1}*

Explanation: This is only an informational message about the Android USB controller external process.

System action: None.

User response: None.

CRRTWM2001E

unexpected exception

Explanation: An exception that could not be handled occurs during processing.

User response: Report exception to product support.

CRRTWM2002E

Enumeration type *{0}* doesn't exist in grammar *{1}*

Explanation: Expected enumeration type is not provided by grammar.

User response: Contact your support.

CRRTWM2003E

IExtendedTypeCodeGen '*{0}*' already defined, at extension id='*{1}*'

Explanation: A mobile extension is registered twice (same or two different with same identifier).

User response: Contact your support.

CRRTWM3001E

Unable to find field *{0}* in object

Explanation: Android object doesn't provide the expected field.

User response: Contact your support.

CRRTWM3002E

Unexpected exception

Explanation: A unexpected exception occurs during processing

User response: Try again, contact your support if problem persist.

CRRTWM4001E

Unable to find field *{0}* in object

Explanation: iOS object doesn't contains the expected field.

User response: Contact your support.

CRRTWM4002E

Unexpected exception

Explanation: A unexpected exception occurs during processing

User response: Try again, contact your support if problem persist.

CRRTWM5001E

Unexpected exception

Explanation: A unexpected exception occurs during processing

User response: Try again, contact your support if problem persist.

CRRTWM6001E

unexpected exception

Explanation: A unexpected exception occurs during processing

System action: None.

User response: Report the exception to your product support.

CRRTWM6002E

cannot load resource '*{0}*'

Explanation: A workbench resource cannot be loaded

System action: Workbench may not work properly.

User response: Close, reopen workbench and try again, if problem persist, contact your support.

CRRTWM6003E

cannot save resource '*{0}*'

Explanation: A workbench resource cannot be saved, some data may be lost if you close workench.

System action: Workbench may not work properly.

User response: Try again, or contact your support.

CRRTWM6004E

cannot create directory '*{0}*'

Explanation: A director cannot be created on you system.

System action: None.

User response: Check if there are enough disk space, or if you have enough right to create directory.

CRRTWM6005E

cannot copy file '*{0}*' to directory '*{1}*'

Explanation: A file cannot be copied to a directory.

System action: None.

User response: Check if there are enough disk space, if directory exists and it's accessible, or if you have enough right to create directory.

CRRTWM6011E

exception thrown while configuring '*{0}*'

Explanation: An exception occurs during the configuration of an element.

System action: None.

User response: Try again.

CRRTWM6012E

exception thrown from component *{0}* while building from '*{1}*'

Explanation: An exception occurs during the build of a mobile application.

System action: Application editor display failed application with error or processing status.

User response: Remove application, retry to add or import application again.

CRRTWM6013I

from component *{0}*: *{1}*

Explanation: A build component display some log entry

System action: None.

User response: Check the entry displayed.

CRRTWM6020W

state machine error thrown during execution

Explanation: Execution engine detect that a invalid state occurs.

System action: None.

User response: Try again or contact your support.

CRRTWM6021E

user error detected during execution: *{0}*

Explanation: This message can also be read in the log entry.

System action: None.

User response: Follow the directions provided in the displayed message.

CRRTWM6030W

device data corrupted during properties update (data: '*{0}*')

Explanation: Mobile client send invalid or corrupted data to workbench.

System action: None.

User response: restart workbench, reboot mobile or contact your support.

CRRTWM7001E

unexpected exception

Explanation: A unexpected exception occurs during processing

User response: Please contact your support

CRRTWM8001E

Unexpected exception while dealing with *{0}*

Explanation: A unexpected exception occurs during processing

User response: Report the exception to your product support.

CRRTWM8010W

Missing translation for key *{0}*

Explanation: A translation key is missed.

User response: Report the exception to your product support.

CRRTWM9010E

error while serving landing page

Explanation: An error has been detected while reading the content of the help landing page in the current installation. The installation might have been altered and is, hence, unreadable.

System action: None.

User response: Reinstall the product.

CRRTWS0001E

An %1 exception occured during translation

Explanation: Exception in execution of Selenium Script.

User response: Verify the Selenium Java Project is open in the workspace and the Selenium Script path referenced in the Compound Test is available in the workspace.

CRRTWS0002I

"ClassName : %1 ProjectName : %2 IsJunit : %3 ExecutionArgs : %4"

Explanation: The Information to the user of the Selenium Script being executed.

User response: None, as this is an information for debugging purposes.

CRRTWS0101W

Exception in opening Script %1 Editor

Explanation: Error in opening the Selenium Java Script from the hyperlink in the Compound Test Editor

User response: Verify if a simple java file can be open from the Package Explorer

CRRTWS0201W

Exception in getting Source (src) folder from Project %1

Explanation: Error in obtaining the Source folder of the project while opening the Selenium Script

User response: Verify the Selenium Java Project has read access and the Selenium Java Project is present is present in the workspace

CRRTWS0202W

Exception in setting Source for IFile : %1

Explanation: Exception in adding a Selenium Test to the Compound Test.

User response: Refresh the Selenium Java Project in the Package Explorer view and Verify if the Selenium Java Project has read access

CRRTWW0001I

Start of Web UI Recorder

CRRTWW0002I

Java Script Request Received

CRRTWW0003I

Java Script Sent in (ms) = %1

CRRTWW0004I

Xhr Request Received

CRRTWW0005I

Xhr Request Process Start

CRRTWW0006I

Xhr Response Time (ms) = %1

CRRTWW0007E

Exception in processing Action

Explanation: An exception occurs while recording an action.

System action: None.

User response: Report exception to product support.

CRRTWW0008E

Exception while sending JavaScript

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0009I

Start of capturing snapshot;

CRRTWW0010I

End of capturing snapshot in (ms) = %1

CRRTWW0011E

Exception in capturing snapshot

Explanation: An exception occurs while capturing the screenshot.

System action: None.

User response: Report exception to product support.

CRRTWW0012E

Exception in sending Xhr response

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0013I

Processing Xhr Request by Recorder.

CRRTWW0014I

Stopping the recorder as browser is closed

CRRTWW0015I

Seems like Browser is Closed.

CRRTWW0016I

Web Application Node Added

CRRTWW0017E

Exception in Recorder Delegate

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0018E

Empty Json Action String. No Operation in Recorder.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0019E

Exception in setting recorder preferences.

Explanation: An error occurred while recording.

System action: None.

User response: Contact __VENDOR_NAME__ Support.

CRRTWW0019I

InWindow Found

CRRTWW0020I

InWindow Added

CRRTWW0021E

Exception in setting Firefox browser preferences.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0022E

Exception in starting Chrome browser.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0023E

Exception in bringing browser to front.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0024E

Exception in starting Internet Explorer.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0025E

Exception in starting Safari.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0026E

Exception in starting Firefox.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0027E

Exception in computing the port from json.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0028E

Exception in serialization.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0029E

Exception in computing the browsers connected to workbench.

Explanation: An internal exception occurs while recording.

System action: None.

User response: Report exception to product support.

CRRTWW0101I

Registered %1 (%2), located at %3

CRRTWW0102I

Found %1 at %2

CRRTWW0103W

Web UI testing is supported on %1 version %2 or later. Found version %3

Explanation: The version of the browser installed in the user's machine is not supported by Web UI.

System action: None.

User response: Install the version of the browser which is supported by Web UI and retry.

CRRTWW0104W

Could not find %1

Explanation: Could not find the browser installed on the machine.

System action: None.

User response: If the user intends to use this browser for playback, then install it and retry. Ignore otherwise.

CRRTWW0105E

Error determining the version of %1

Explanation: An exception occurs while trying to find the version of the browser.

System action: None.

User response: Report this problem to the product support.

CRRTWW0106E

There was a problem adding the browser %1

Explanation: An exception occurs while adding the browser to the wizard.

System action: None.

User response: Report exception to product support.

CRRTWW0107E

There was a problem setting the Web UI playback preference

Explanation: An exception occurs while setting the Web UI playback preference.

System action: None.

User response: Report exception to product support.

CRRTWW0109E

There was an exception.

Explanation: An exception occurs in the UI.

System action: None.

User response: Report exception to product support.

CRRTWW0110E

There was a problem scanning and marking the test.

Explanation: An exception occurs while scanning and marking the test.

System action: None

User response: Report exception to product support.

CRRTWW0140I

Register Browsers

CRRTWW0150I

Hybrid, invoking action using WebDriver

CRRTWW0151I

Script Method : %1

CRRTWW0152I

Invoke action "%1" on "%2"

CRRTWW0153I

Hybrid, invoking JavaScript

CRRTWW0154I

Start %1 using %2

CRRTWW0155I

Done with execution.

CRRTWW0156I

Creating xpath for %1

CRRTWW0157I

Generated XPath %1

CRRTWW0158I

Searching by %1

CRRTWW0159I

Execute : %1

CRRTWW0160I

Found %1 objects

CRRTWW0161I

Time taken (%1) = %2

CRRTWW0162I

Switched to window : %1

CRRTWW0163I

Return value from JS : %1

CRRTWW0200E

Exception : %1

Explanation: An unexpected exception occurs during playback.

System action: None.

User response: Report exception to product support.

CRRTWW0201E

%1 is not implemented.

Explanation: An action specified in the test is not implemented in playback.

System action: None.

User response: Report this problem to the product support.

CRRTWW0202E

Unexpected return value from JS, %1

Explanation: An unexpected value is found during playback.

System action: None.

User response: Report this problem to the product support.

CRRTWW0203W

Unable to delete file

Explanation: An unexpected exception occurs during playback.

System action: None.

User response: Report exception to product support.

CRRTWW0300W

Failure to create directory during test generation.

Explanation: An exception occurs while generating a Web UI test.

System action: None.

User response: Report exception to product support.

HOTC0009E

License server connection test failed: %1

Explanation: A license server connection test failed.

System action: License server details cannot be displayed.

User response: Refer to the licensing error message for more details.

RMSE0003W

RMSE0003W There are currently no selected counters for the source named *{0}*.

Explanation: The source has no counters selected.

System action: Execution of the schedule will continue but the information related to this source won't be collected.

User response: Consider selecting at least one counter from the Resource Monitoring Service web console.

RMSE0004W

RMSE0004W The source named *{0}* is no longer available.

Explanation: This source has been removed from the Service web console after it was added to this schedule.

System action: Execution of the schedule will continue but the information related to this source won't be collected.

User response: Consider adding it back, then edit the schedule to update the sources to be monitored during its execution.

RMSE0005W

RMSE0005W The source named *{0}* is reporting the error message *{1}*.

Explanation: Look at the reported error.

System action: Execution of the schedule will continue but the information related to this source won't be collected.

User response: Consider fixing it from the Resource Monitoring Service web console.

Frequently Asked Questions

This document answers some of the common queries in mobile testing.

Questions

- **Q 1.** Why am I unable to connect my device to the workbench? on page 1538
- **Q 2.** Why are some of the UI elements or actions are not captured by Rational® Test WorkbenchHCL OneTest™ Studio? on page 1538

Answers

- **A 1.** Make sure that the device and test workbench are in the same network.
- **A 2.** The UI elements or actions might not be supported by the product. You can manually add an action to the test script. To add the action using selected graphical elements, see Adding user actions in a test from the SmartShot View on page 596.

Troubleshooting in the Functional Test perspective

In this section, you will learn how to troubleshoot the tests in the Functional Test perspective.

Troubleshooting functional tests in Mozilla Firefox browsers

If you encounter problems while testing in Mozilla Firefox browsers, the following workarounds might help resolve them.

Note: The following issues and workarounds are applicable to Rational® Functional TesterHCL OneTest™ UI 8.5.1 9.1 and above and Firefox 18 and above on Windows™ computers.

It is not possible to record on a Firefox browser

This problem could occur for the following reasons:

- The browser was not properly enabled. To ensure that the browser is properly enabled, in Firefox, click **Tools** > **Add-ons** > **Extensions**, and verify that the Rational® Functional TesterHCL OneTest™ UI Firefox Enabler is present and enabled.
- The Mozilla Firefox browser is associated with JRE 1.6. Ensure that your test environment has JRE 1.7, which is enabled and associated with the Firefox browser. On your Windows™ computer, this option can be controlled from **Control Panel** > **Java** (Choose Java™ 1.7) > **Advanced** > **Default Java for Browsers**> **Mozilla Family**.
- JavaScript™ is not enabled on your browser. Navigate to **Firefox** > **Options** > **Content** and select the **Enable JavaScript** box.
- The web page that you are trying to run is available on the local file system. Host the web page on a web server and then try recording. In Firefox, pages that are loaded from local file systems can be tested.
- The browser was started with a blank home page that is without a home URL. Always specify a home URL for the browser.
- The Firefox enabler extension that is installed is not the correct extension for your browser version. If you uninstall a newer version of Firefox and install an earlier version, the extension that supports testing on Firefox is not compatible with the earlier version. To ensure that you have the correct extension when you are changing versions, disable Firefox from the **Enablement Wizard**, and then re-enable it. Re-enabling Firefox installs the proper extension for your browser version.
- The port number that is specified in the Firefox enabler extension options is not the same as the specified the port number specified in Rational® Functional TesterHCL OneTest™ UI Webserver Configuration preferences. To verify the port number in Rational® Functional TesterHCL OneTest™ UI, select **Window** > **Preferences** > **Functional Test** > **Webserver Configuration**.
- For Firefox version 18 and above, when a document is loaded, the Rational® Functional TesterHCL OneTest™ UI **Firefox Enabler** loads an applet that enables communication with Rational® Functional TesterHCL OneTest™ UI. In some instances, you see an initializing screen indefinitely. This issue is most likely because the applet cannot be validated without an internet connection. To address this issue, online validation must be disabled. To disable online validation, open the **Java Control Panel**. In the **Advanced** tab, under Perform certificate revocation checks on" select the Do not check (not recommended) option. Click **Apply** save the changes and then click **Ok** to restart the browser. Rational® Functional TesterHCL OneTest™ UI uses applets

that are signed with secure certificates in order to comply with Java™ security features. When prompted by the browser, accept to run applet permanently to ensure Rational® Functional TesterHCL OneTest™ UI functions properly. Also, ensure that the proper delays are in place for playback.

While recording on Mozilla Firefox browsers, some dialog boxes are recordable while others are not

Rational® Functional TesterHCL OneTest™ UI supports the following dialog boxes:

- Dialog boxes that are supported by Frameworks. Example: Dojo dialog box.
- Native or XUL dialog boxes

Rational® Functional TesterHCL OneTest™ UI does not support the following dialog boxes:

- Recording on JavaScript™ alert boxes is not supported.

Unable to test applications in Linux® with Rational® Functional TesterHCL OneTest™ UI on Mozilla Firefox browsers

Rational® Functional TesterHCL OneTest™ UI supports testing applications on Linux® with Firefox through Mozilla Firefox ESR version 17 and later releases of 17.

Unable to use Ajax related APIs on an Ajax application that is running on Mozilla Firefox browsers

Record and playback on Ajax-based actions work as expected. However, for Firefox version 18 and above, the following Ajax related APIs are not supported.

- waitForAjaxCompleteRequests
- waitForAjaxPendingRequests
- setAjaxTrace
- getAjaxPendingRequests
- getAjaxCompletedRequests

To introduce delays, use the sleep API. For more information, see AJAX support on page 1708.

Unable to test embedded PDF files in Firefox browsers

In Rational® Functional TesterHCL OneTest™ UI 8.5.19.1, embedded PDF files in Firefox 18 and above are supported only when the PDF file is present in the browser along with other HTML controls. For example, the Recorder and Player will not recognize the PDF controls correctly if the file was opened by right-clicking the document and selecting **Open with** > **Firefox**.

Unable to launch the Verification Point Comparator from execution logs generated through Rational® Quality Manager

In the playback log, the ComparatorApplet,which is responsible for launching the Verification Points Comparator is hosted on a local server started by Rational® Functional TesterHCL OneTest™ UI or one of its client processes. In the absence of this server, the comparator does not launch. To address this issue, ensure that Rational® Functional TesterHCL OneTest™ UI is running on the computer where playback logs from Rational® Quality Manager are being viewed.

Unable to launch the Verification Point Comparator from playback logs on Linux® installations

On Linux® installations, the Verification Point Comparator cannot be opened through the playback logs. Instead, open the comparator from the project's logs folder (`<projectname>_logs`).

Unable to play a script that was recorded on a stand-alone PDF document in Firefox on an embedded PDF document

In Firefox version 19 and above, the plugin pdf.js is used to render PDF documents. This plugin renders PDF documents as HTML pages, thus Rational® Functional TesterHCL OneTest™ UI records the controls as HTML. For a script recorded on a stand-alone PDF document to be compatible on an embedded PDF document in Firefox, you can disable the pdf.js plugin by typing `about:config` in the address. When prompted, click the **I'll be careful, I promise!!** button. Search for the pdfjs.disabled flag and right-click then select **Toggle** to change the value from **false** to **true**. Restart Firefox to apply the changes.

Unable to playback scripts on listbox controls when Firefox is maximized

During playback on Firefox, Rational® Functional TesterHCL OneTest™ UI sometimes fails to click listbox controls when the browser is maximized. In order to correct this, you can adjust the zoom level of the browser or run the playback with the browser window not maximized.

Related information

AJAX support on page 1708

HTML support on page 1715

Unable to test eclipse-based applications

Generally the applications that are enabled for functional testing is in the enabled state after you upgrade Rational® Functional TesterHCL OneTest™ UI. If you find any issues while testing the applications, disable the Eclipse application and enable it again for functional testing.

Disabling eclipse-based applications:

If you enabled the Eclipse application using the Rational® Functional TesterHCL OneTest™ UI enable applications option, disable the application from the Enable Applications window in Rational® Functional TesterHCL OneTest™ UI.

To disable the Eclipse application that was enabled using the Eclipse Software Updates feature:

1. Open the application under test.
2. Click **Help > Software Updates**.
3. Click the **Installed Software** tab.
4. Select the Eclipse application and click **Uninstall**.

Ambiguous object recognition in functional testing

Ambiguous recognition occurs when Rational® Functional TesterHCL OneTest™ UI can not uniquely identify an object in the system-under-test. This commonly happens when Rational® Functional TesterHCL OneTest™ UI cannot differentiate between an instance of the application-under-test started by a script playback and an instance of the same application inadvertently left open previous to script playback. This also applies to identical windows from one application and identical HTML documents. Ambiguous recognition will cause script playback failure unless the duplicate application is closed.

If Rational® Functional TesterHCL OneTest™ UI finds more than one instance of the application-under-test during the playback of a script the **Ambiguous Recognition** window will open allowing you to close the duplicate instance and resume playback.

Preventing ambiguous recognition

One common cause of ambiguous recognition is residual windows left open from a previous playback of a test script.

To avoid this issue take the following actions:

- Make closing the application-under-test the last action recorded in the test script.
- If script playback fails, close all windows opened by script playback before replaying the script.

Dealing with ambiguous recognition

If the **Ambiguous Recognition** window opens correct the situation and restart playback.

The **Ambiguous Recognition** window opens and playback pauses.

1. Minimize open windows until the **Ambiguous Recognition** window is visible.
2. Find and close the duplicate application instance using the information in the **Ambiguous Recognition** window.
3. Click **OK** in the **Ambiguous Recognition** window to resume playback.

Screen snapshot on playback failure of functional tests

If playback of a script causes an exception to be thrown, Rational® Functional TesterHCL OneTest™ UI takes a screen snapshot at the time of the failure. The screen snapshot is accessible through the log.

Accessing the screen snapshot in an HTML log type

HTML is the preferred log type to access the screen snapshot.

Select **HTML** as the log type in the Logging Preferences Page in FT Java™ or the Logging Options Page in FT .Net.

After playback fails the log opens in your browser.

1. Find the screen snapshot image near the bottom of the log.
    - Click the image or link to view full size.
    - Right click to save, print, or email the JPEG image.

Taking a screen snapshot with scripting

RootTestObject exposes a getScreenSnapshot method that will return a snapshot of the screen. GuiTestObject exposes the same method, but only captures the portion of the screen rendering the TestObject. LogInfo, LogError, and LogWarning all have overloads that will take a snapshot and add it to the log.

Tips and tricks for functional testing HTML applications

This topic provides tips and tricks for recording and playing back scripts to test HTML applications.

Start recording first and then start the application

When recording scripts on your HTML applications, useRational® Functional TesterHCL OneTest™ UI to start the application during recording.Rational® Functional TesterHCL OneTest™ UI opens the HTML page that you specify in your default browser or in a specific browser.

Recording a hover on HTML menus

When recording scripts on your HTML applications, you can record a hover on drop-down menus that are activated when you roll the mouse over the drop-down menu. These drop-down menus are implemented with DIV tags. To record a hover for drop-down menus, and make the sub-menu drop down, hover the mouse over the menu item text and press Shift. Make sure the mouse is on the text of the menu item and not on the blank space to the right of the menu item text.

Use deleteCookies method in your scripts

Two versions of the deleteCookies method are available. One method deletes all cookies for the current profile or user and the other method deletes cookies in a specific path or domain for the current profile or user. For information, see theRational® Functional TesterHCL OneTest™ UI API Reference, in the com.rational.test.ft.object.interfaces package, under IBrowserObject. See https://www.ibm.com/support/knowledgecenter/SSJMXE_8.6.0/com.rational.test.ft.api.help/ApiReference/com/rational/test/ft/object/interfaces/IBrowserObject.html?lang=pl.

Avoid including menu items in scripts

Because selections on browser menus are recorded based on their screen coordinates, scripts may not play back reliably if the browser size or position change. Also, menus are different on different browsers, which may also cause scripts to play back incorrectly.

Make sure Java applets are in full view during playback

If you resize the browser to a smaller size,Rational® Functional TesterHCL OneTest™ UI does not scroll the applet objects into view during playback if they are not in view.

Use the loadURL( ) method to change URLs

The location of the **Address** field in a browser is based on screen coordinates, which can change if the browser's size and position change. A script usually fails if you click in the **Address** field and type the new URL. When recording, insert a browser click (Browser_htmlBrowser) in your script to change URLs.

1. When recording, click any empty space in the browser header to include a browser click in your test object map.
2. After recording, view the script and place the cursor on a blank line in the script.
3. In the Script Explorer, expand **Test Objects**, right-click **Browser_htmlBrowser**, and select **Insert Asset at Cursor**.
4. Select the `loadURL(String)`method.
5. Type the name of the new URL between the parentheses of the `loadURL` statement.
6. Insert a semicolon (;) at the end of the line for only Java™, and not for VB.NET.

Using .size property for INPUT elements

If you use .size Property for INPUT elements and do not specify the .size property within the Html of an INPUT element, the default value returned by the Internet Explorer is 20.

Use toolbar buttons common to both browsers

When you create a cross-platform script, avoid toolbar buttons that only appear in one browser. The following toolbar buttons are common to both browsers:

- Back
- Bookmarks /Favorites (Internet Explorer)
- Close
- Forward
- Home
- Maximize
- Minimize
- Search (button only)
- Stop

Use the close button to exit a browser

The Close button is available in the Internet Explorer. When you record a cross-platform script, avoid using alternative methods of exiting the browser. For example, pressing Alt+F+C works only for Internet Explorer. Either key combination causes a script to fail when run on the other browser.

Check the .readystate of the browser object

Sometimes script playback for testing HTML application fails if the ready state of the browser object is not 4. Ensure that the ready state of the browser is 4 while playing back a script. You can do this by modifying the test script manually as shown in the examples.

Script to check the browser state in Java™: `logInfo("Ready State #: "+browser_htmlBrowser().getProperty(".readyState").toString());`

Script to check the browser state in .Net: `LogInfo("Ready State #: " & Browser_HtmlBrowser().GetProperty(".readyState").ToString)`

Use waitForExistence method to compensate for browser startup speed

Use a waitForExistence method when recording cross-browser scripts to wait for a browser. For example:

1. When recording, start the application.
2. Click the **Insert Verification Point or Action Command** button  on the Recording toolbar.
3. In the **Select an Object** page of the Verification Point and Action Wizard, click the **Object Finder** icon  and drag it over the HTML page (not the browser itself).
4. Click **Next**.
5. In the **Select an Action** page of the Verification Point and Action Wizard, click the **Wait for Selected TestObject** option.
6. If necessary, clear **Use the defaults** to change the **Maximum Wait Time** and **Check Interval** settings, which are 2 minutes and 2 seconds, respectively.
7. Click **Finish**.

Avoid these click and key combinations in cross-platform scripts

To handle a pop-up menu, some browsers ignore a click action on a link immediately following a right-click. When this click combination is necessary, right-click the link, click an empty space in the document, and then click the link.

In some browsers, pressing Ctrl and clicking a link opens the page in another instance of the browser. This same key sequence results in a normal link click in other browsers. A script that contains this combination of actions plays back differently and should be avoided for cross-browser testing.

Run a utility to fix badly formed HTML

Occasionally errors in HTML can cause different browsers to interpret the HTML DOM hierarchy differently. A script that runs successfully in one browser can fail in another. Record one script against each browser and compare the

resulting test object maps. If the maps show a different hierarchy, run a utility, such as HTML Tidy. If the utility reports errors, it is possible that the errors are causing the different interpretations of the object model, resulting in different hierarchies. HTML Tidy is available from the World Wide Web Consortium, www.w3.org.

Handling pop-up message boxes

When recording a script in some browsers, a pop-up message (browser user interface dialog boxes), such as encryption notices occasionally appear. When recording a cross-browser script, you do not want to include these message boxes, because they may not appear in other browsers. To avoid this problem:

1. When a pop-up message appears, pause recording.
2. Select any check box on the message that prevents the message from appearing again.
3. Click **Cancel** to close the message box.
4. Resume recording.

You can modify your script to handle these kinds of message boxes, but the code can be complicated. For more information, see the ExtendingRational® Functional TesterHCL OneTest™ UI functionality topics.

When you record a cross-browser compatible (a script that is compatible across all browsers thatRational® Functional TesterHCL OneTest™ UI supports) script, try to avoid recording any pop-up message boxes. If you are not recording a cross-browser compatible script, you can record pop-up message boxes in your script.

Rational® Functional TesterHCL OneTest™ UI supports the Login, File Download, Certificate/Security Warnings, File Picker (File Open/File Save), and Print dialog boxes on the Windows® platform. These user interface dialog boxes are for a specific browser and are not cross-browser compatible. In most cases, the Login dialog box is cross-browser compatible.

Testing URLs without configuring the application

When you configure an application, Functional Test adds the application name to the Application Configuration Tool. If you test a lot of different URLs, the Applications list can become long. If you do not want to add a URL to the list, you can use the startBrowser command in an empty script to test it.

1. Create a new functional test script without recording.
2. On a blank line, type the following command:

   ```
   startBrowser ("url");
   ```

3. Save the script and run it.
4. When the page is displayed, on the Functional Test toolbar, click **Insert Recording into Active Functional Test Script** and start recording against the page.

Testing HTAs

Rational® Functional TesterHCL OneTest™ UI supports testing Microsoft® HTML Applications (MSHTA). Before you can test a MSHTA, you must configure it by running mshta.exe. To configure each HTA you want to test:

1. In the **Kind** field of the Application Configuration Tool, select **executable** or **batch**.
2. In the **Executable file** field, select **mshta.exe**.
3. In the **Args** field, pass the parameter *x*`.hta` to the executable, where *x* is the name of the HTA file.

For more information, see Configuring Applications for Testing.

Handling Java plug-in errors

If an error about the Java™ plug-in is returned, when you test HTML applications or start the Comparator from the **View Results** link in the HTML log, you need to verify that your browser's Java™ plug-in is configured properly. For instructions, see the related topic about enabling the Java plug-in of a browser.

Note: To view the Rational TestManager version that can be integrated with Rational® Functional TesterHCL OneTest™ UI, see List of supported domains for functional testing by releases of Rational® Functional TesterHCL OneTest™ UI tech. note.

Java applets in HTML pages

You can test Java™ applets within a browser (Firefox, Internet Explorer). Java™ applets are not mapped as nested within HTML but are recorded as top-level objects. In the test object map, applets appear at the top level.

If the object cannot be found by "The Java™ Test Domain", the HTML Applet Test Object (HTML AppletProxy) is used as the fall-back test object, which provides only coordinate-based recording.

Requirements for testing applets within a browser

- ◦ The Sun Java™ Plug-in is required for running and testing applets.
  ◦ To use Java™ applets with Firefox, Java™ 2 Standard Edition Runtime version 1.4 or greater is required, and the associated Java™ Plug-in must be installed.
- **Internet Explorer**
  ◦ The Sun Java™ Plug-in is not required to run applets, but it is required for testing applets with Rational® Functional TesterHCL OneTest™ UI. If the Java™ Plug-in is not installed, the Microsoft® JVM is used to run applets, and Rational® Functional TesterHCL OneTest™ UI is not designed to enable the Microsoft® JVM.
  ◦ If you want to use a Java™ Plugin older than 1.4 with Internet Explorer, you must turn off Applet Support:
    1. From the Windows® **Start** menu, run regedit.
    2. Open HKEY_LOCAL_MACHINE\Software\Rational Software\Rational Test\8 HKEY_LOCAL_MACHINE\Software\HCL Technologies\HCL OneTest UI.
    3. In the right pane, right-click and click **New > String Value**.

4. Set the name of the new string to `Applet Support`.
5. Double-click the newly created string.
6. In the **Value data** field of the Edit String dialog box, type `0`.
7. Restart your computer.

- **Rational® Functional TesterHCL OneTest™ UI**
  - For Internet Explorer, use Java™ Plug-in version 1.4 greater. Earlier versions of the Java™ Plug-in, including 1.2.2, and 1.3.1_01 do not work with Rational® Functional TesterHCL OneTest™ UI.
  - You must enable the JVM (JRE) that the Java™ Plug-in is using. When a JavaSoft JRE or JVM is installed, it may install a Java™ Plug-in also. If so, you must use Rational® Functional TesterHCL OneTest™ UI to enable the JVM used by the Java™ Plug-in. For information, see Enabling Java Environments on page 778.
  - Rational® Functional TesterHCL OneTest™ UI uses the most recently installed Java™ Plug-in/JRE. If an unsupported Java™ Plug-in is installed (for example, Version 1.2.2 in Internet Explorer), Rational® Functional TesterHCL OneTest™ UI stops working with the browser.
  - The Java™ Plug-in uses its default JRE (the JRE with the same version as the plug-in), unless specified otherwise. You can change the default JRE in the Java™ Plug-in control panel application.
  - Rational® Functional TesterHCL OneTest™ UI attempts to locate the most recently installed Java™ Plug-in and enable its default JVM.
  - **Java™ Applets in HTML**
    - An applet can be specified in HTML using an APPLET tag, an OBJECT tag, or an EMBED tag.
    - For Internet Explorer, until version 1.3 of the Java™ Plug-in, the OBJECT tag had to be used to specify the use of the Sun JVM for applets. In version 1.4 and later, during installation of the Java™ Plug-in, the use of the Java™ Plug-in/JRE may be selected as the default for Internet Explorer (APPLET tags), allowing both APPLET and OBJECT tags to be used.
    - A Java™ Plug-in HTML Converter is available from Sun Microsystems to convert APPLET tags to a set of OBJECT and EMBED tags within the HTML document.
    - Make sure Java™ applets are visible during playback. If you resize the browser to a smaller size, Rational® Functional TesterHCL OneTest™ UI does not scroll the applet objects into view during playback.

Standard properties available for functional testing HTML objects

Standard properties provide a common way to access properties and their values across browsers. This topic lists the standard properties available for HTML objects.

Most of these properties are modeled on HTML element attributes defined by the W3C.

Property	Use
.align	Value of the **align** attribute of the element. Valid values are bottom, middle, and top.
.alt	Value of the **alt** attribute of an element. This is the "alternate" text for the element, usually displayed by the browser when the mouse hovers over the element.
.border	Value of the **border** attribute of the element. Returns the number of pixels.
.bounds	Rectangle that represents the bounding rectangle of the object in screen coordinates.
.caption	For TABLE elements, the value of the **caption** attribute. For an HTML dialog box, this is the name of the dialog box.
.cellIndex	Cell index of an element with respect to its row (>=0).
.checked	Boolean value that indicates whether a check box is checked (true) or not (false).
.class	TestObject class name; for example, "HtmlTable" for a TABLE element.
.class-Name	Value of the class attribute of an element (used for stylesheets in HTML).
.clientRect	Bounding rectangle of the element in client coordinates.
.code	Value of the **code** attribute of an APPLET element.
.codeBase	Value of the **codeBase** attribute of an APPLET element.
.colSpan	Value of the **colSpan** attribute of an element.
.cookie	Current value of the cookie for the document.
.coords	Value of the **coords** attribute of an element. This is a string containing the coordinates used to define the AREA element of a client-side image map. In the form x1, y1, x2, y2, and so on.
.default-Checked	Boolean value for the **defaultChecked** attribute of the element.
.defaultS-elected	Boolean value; when true indicates that the OPTION element in a SELECT element (listbox or drop-down) is selected by default when the page is displayed.
.defaultVal-ue	Value of the **defaultValue** attribute of the element.
.disabled	Value of the **disabled** attribute of an element, returned as a boolean. If true, user input is currently disabled for this item.
.hasFocus	Indicates whether the current element has focus.
.hasScript	Boolean value; true when a script has been associated with actions on this element.
.height	Value of the **height** attribute of an element. For an Image element, this is the display height in pixels for the image.

Property	Use
.href	Value of the **href** attribute of an element. This is a URL used by ANCHOR and AREA elements to indicate the result of clicking the corresponding element.
.hspace	Value of the **hspace** attribute of an element, the amount of whitespace inserted to the left or right of an IMG, OBJECT, or APPLET element.
.id	Value of the **id** attribute of an element.
.indeterminate	Boolean value; true when the status of the check box has been changed.
.index	Index of the OPTION element within a listbox or combodropdown list.
.isMap	Boolean value; for Image elements (IMG), this value is true when the element is a server-side image map.
.length	Value of the **length** attribute of an element. For a SELECT element, this indicates the number of items in the list.
.maxLength	For an edit control (Input type=Text or TextArea) this indicates the maximum number of characters a user can enter.
.multiple	Boolean value; for a SELECT element (listbox or combo dropdown), a value of true indicates that the list supports multiple selections.
.name	Value of the **name** attribute (Form elements and Frames only).
.noHref	Value of the **noHref** attribute of an element. When set on an AREA element, indicates that the corresponding area has no associated action.
.offsetHeight	Height of the element.
.offsetLeft	Offset, in pixels, of the element from its left edge to the left edge of its parent element in the DOM.
.offsetRight	Offset, in pixels, of the element from its right edge to the right edge of its parent element in the DOM.
.offsetTop	Offset of the element from the offset of its parent element in the DOM.
.offsetWidth	Width of the element.
.readOnly	Value of the **readOnly** attribute of an element. Boolean value; true when the form element is read-only.
.readyState	Current status of a browser, indicating whether it is currently loading a document or ready for user input. This is an integer value: 0 - Uninitialized 1 - Loading 2 - Loaded

Property	Use
	3- Interactive 4 - Complete (ready)
.rowIndex	Row index of an element in a table (>=0).
.rows	Value of the **rows** attribute of a TEXTAREA element, indicating the size of the edit control in the number of rows of text.
.rowSpan	Value of the **rowSpan** attribute of an element.
.screenLeft	Upper left corner of bounding rectangle in screen coordinates, x component.
.screenTop	Upper left corner of bounding rectangle in screen coordinates, y component.
.select	Boolean value; true when the FORM element is highlighted to receive user input.
.selected	Boolean value; true when the OPTION element in a SELECT element (listbox or dropdown) is selected.
.selectedIndex	Value of the **selectedIndex** attribute of an element. For a single selection Select element, this indicates which option element is selected. Integer in the range of >=0.
.shape	Value of the **shape** attribute of an element. Used for AREA elements in client-side image maps. Valid values are default, rect, circle, and poly.
.size	Value of the **size** attribute of an element. For a Select element, the number of items displayed at one time in the list. If size > 1, the list appears as listbox; otherwise the list appears as a Combodropdown.
.src	Value of the **src** attribute for the element. For images and image buttons, this is a URL specifying the image file.
.tag	HTML tag for the element.
.target	Value of the **target** attribute of an element. For anchors, this indicates the name of the target frame, that is, the frame where the document should be opened.
.text	Text inside of the HTML tags for bounding the element. For example: `<A>This is an Anchor<IMG src=map.gif></a>` The text property returns "This is an Anchor." If consecutive white space characters are found, all white spaces are combined and reduced to a single blank character.
.title	Value of the **title** attribute of an element. This is frequently the text displayed when hovering over the element with the mouse.
.type	Value of the **type** attribute of an element. For example, for an Input element this is text, password, check box, radio, submit, image, reset, button, hidden, or file.
.url	URL of the document.

Property	Use
.useMap	Value of the **useMap** attribute of an element. The value is a string specifying a URL and is used for IMAGE elements (IMG) to indicate a client-side image map. The URL points to the map associated MAP element. Frequently, this is a document-relative reference.
.value	Value of the **value** attribute of an element. In Form elements this represents the value sent when the form is posted.
.vspace	Value of the **vspace** attribute of an element, the amount of whitespace inserted to the above or below an IMG, OBJECT, or APPLET element.
.width	Value of the **width** attribute of an element. For an IMAGE element, this is the display width in pixels for the image.
.window	Heavyweight window for the element returned as a long. For the Browser, this is the top-level window.

Uninstalling Rational® Functional TesterHCL OneTest™ UI cleanly

If you have any issues during uninstall and reinstall of Rational® Functional TesterHCL OneTest™ UI, you can perform a few tasks to verify whether the required processes are stopped and all the files are deleted from the computer.

1. To uninstall the packages, you must log in to the system using the same user account that you used to install the product packages.
2. Before you uninstall Rational® Functional TesterHCL OneTest™ UI, close the Eclipse and Visual Studio IDEs, as well as any open web browsers, and all other applications that are enabled by Rational® Functional TesterHCL OneTest™ UI. To ensure that all the processes have stopped, you can use any of the following tools:
    - Use the Task Manager to kill all the Rational® Functional TesterHCL OneTest™ UI processes such as java.exe and javaw.exe.
    - You can use Process Explorer from Microsoft to search and stop all the Rational® Functional TesterHCL OneTest™ UI processes.
        a. In the Process Explorer, click **Find > Find Handle DLL**.
        b. Type `rtx` in the **Handle** or **DLL substring** field.
        c. Kill all the processes that are listed in the **Process Explorer Search** window.
3. Uninstall Rational® Functional TesterHCL OneTest™ UI using the Installation Manager.
4. After uninstalling Rational® Functional TesterHCL OneTest™ UI, verify if the uninstallation process has deleted the assembly entries.
    a. Click **Start > Run** and type assembly.
    b. Delete the following assembly instances if they still exist:

- rtxftnet
- policy.7.0.rtxftnet

Note: If you are unable to delete the assembly entries, open the Windows Registry editor and search for the assembly. Delete the entries from the Registry and then try to delete the assembly instance.

5. Click **Start > Run** and type `regedit` to open the registry editor. Expand **HKEY_LOCAL_MACHINE > SOFTWARE > Rational Software > Rational Test** and delete the **8** folder.
6. Click **Start > Run** and type `regedit` to open the registry editor. Expand **HKEY_LOCAL_MACHINE > SOFTWARE > HCL Technologies > HCL OneTest UI >** and delete the **HCL FT JRE** folder.
7. Delete all the files and folders in the product installation directory if they still exist. For example, `C:\Program Files\IBM\SDP` `C:\Program Files\HCL\HCLOneTest`.
8. To delete the configuration and customization files, delete the `RFT OneTestUI` folder that is available by default at `C:\ProgramData\IBM` `C:\ProgramData\HCL` location.
9. To delete the user preference settings of Rational® Functional TesterHCL OneTest™ UI, delete the `RFT OneTestUI` folder that is available by default at `C:\Users\<user name>\AppData\Roaming\IBM` `C:\Users\<user name>\AppData\Roaming\HCL` location.
10. To delete the workspace data, delete the `rft_private_workspaceOneTestUI_private_workspace` folder that is available at `C:\Users\<user name>\IBM\rationalsdp` `C:\Users\<user name>\HCL\HCLOneTest\workspace` location if it still exists.

Problems with object recognition

If you encounter a problem with object recognition during testing, you might be able to resolve the problem by following these instructions.

These issues might occur during object recognition:

- Objects and controls are not recognized on page 1553
- The click action on UI controls do not work on page 1554
- The playback of a context menu fails in Linux operating systems on page 1554
- Previously recorded scripts do not work on page 1555

Objects and controls are not recognized

Problem

Some controls in an application are not recorded in the same way that other objects from the same domain are recorded. This is due to one of the following reasons:

- The controls are custom controls that Rational® Functional TesterHCL OneTest™ UI does not officially support. Therefore, the controls are recorded in a generic domain, as shown in this example::

```
afxWnd90uwindow().click(atPoint(252,212));
```

- The environment for the domain was not configured correctly, as in the case of Flex applications. For example, actions on controls in Flex domains that are configured incorrectly are recognized as follows:

```
oleObjectactiveXControl2().click(atPoint(102,10));
```

Resolution

To resolve this problem, complete these steps:

1. Create a custom proxy for those controls or submit an enhancement request to get your application supported by Rational® Functional TesterHCL OneTest™ UI.
2. Correctly configure the domain by completing the instructions in Flex applications testing process on page 809.

The click action on UI controls do not work

Problem

When you run a functional test by using the Web UI engine, the test steps that contained the 'click' action might fail as the UI control is not correctly recognized. This problem might occur because of the changes in the settings of the async flag in the `ivory.properties` file.

Resolution

You must set the async flag to *false* to resolve the issue:

1. Open the `ivory.properties` file located at `<Product installation directory>\IBM\SDP\FunctionalTester\bin` `<Product installation directory>\HCL\HCLOneTest\FunctionalTester\bin`.
2. Set `functional.tester.html.playback.async=false`.
3. Restart Rational® Functional TesterHCL OneTest™ UI.

The playback of a context menu fails in Linux operating systems

Problem

When you record a functional test in Windows and play it back in Linux, the test steps that contained the contextMenu.click() action might fail with an *object not found* error as the UI control is not correctly recognized.

Resolution

As a workaround, you must edit the test and use the following alternatives to the contextMenu.click()action in your test script:

- Different key combinations to navigate and select the context menu options in the application under test (AUT). For exmaple, you can use alt+enter or alt+shift+S as a key combination to replace the contextMenu.click() action in the test script.
- Alternate menu items to complete the action in the application under test (AUT).

Previously recorded scripts do not work

Problem

The test object map lists the test objects in the application under test, in a hierarchy. If the application under test changes, the object hierarchy might change. During playback, Rational® Functional TesterHCL OneTest™ UI cannot find an object whose hierarchical position changed, and as a result, playback fails.

Resolution

1. Use the dynamic find feature. When a search that is based on object recognition scoring (ScriptAssure) fails to find objects whose hierarchy has changed, the dynamic find feature searches for such objects. To convert an object to a dynamic object:
    a. From the test object map menu, right-click the test object map, and click **Convert To Dynamic Test Object ()**.
    b. In the **Convert To Dynamic Test Object** window, select **Select the parent to anchor in the object hierarchy**. The new object becomes a descendant of its parent.
    c. Select the object to convert, and click **Finish**.
2. Search for the object by using the find method instead of the Record-Playback method. When you use `find( )`, Rational® Functional TesterHCL OneTest™ UI searches for a matching object in the entire hierarchy. Therefore, a change in the hierarchy does not cause the playback failure.

Unable to find the correct control by using the Insert Test Object option

When you try to add a step in the functional test script by using the Insert Test Object option, Rational® Functional TesterHCL OneTest™ UI fails to find the correct control.

Cause

Rational® Functional TesterHCL OneTest™ UI fails to find and highlight the correct control with a red rectangle if the display scaling is incorrect, and if some settings in the Microsoft Edge or Google Chrome browser are not suitable for the AUT.

Resolution

1. Ensure that the display scaling of the system always remains at 100%.
2. Open the browser settings to disable the options that support the apps and processes running in the background when the browser is closed.

    Go to step 3 on page 1556 for Edge or go to step 4 on page 1556 for Chrome.

3. Disable the following options on the **System and performance** page in Edge:
    - **Startup boost**
    - **Continue running background extensions and apps when Microsoft Edge is closed**
4. Disable the **Continue running background apps when Google Chrome is closed** option on the **System** page in Chrome.
5. Restart the browser and open the application for testing.
6. Add steps in the functional test by using the **Insert Test Object** option.

Note: If the problem persists, contact the support team.

Unable to create a verification point by using the vpDynamic action

When you run a functional test for the first time to create a verification point by using the vpDynamic("vpName").performTest() action in the test script, the test step fails.

Resolution

1. Go to **Window > Preferences > Functional Test > Recorder**, and then clear the **Record Test Object Relative Verification Point** checkbox.
2. Play back the functional test, and then create a verification point.
3. Go to **Window > Preferences > Functional Test > Recorder**, and then select the **Record Test Object Relative Verification Point** checkbox.

   You must select this checkbox for any subsequent test runs.
4. Play back the functional test to verify the newly created verification point.

Troubleshooting issues in SAP tests

You can find information about the issues or problems that you might encounter while you test the SAP GUI. Details about issues, their causes and the resolutions that you can apply to fix the issues are described in the following table.

Problem	Description	Solution
The controls on the **SAP Logon** window are not recorded.	While you record the controls on the **SAP Logon** window, either the controls are not captured or the recognition of the controls is very slow.	You must first select the configured server name in the **Connections** column, and then click **Logon**.
The **Close** icon at the upper-right corner of the **SAP Logon** window is not recorded.	The action performed on the **Close** icon on the **SAP Logon** window is not recorded. Even though if it is recorded correctly, the playback of the action on the **Close** icon fails.	While you record, you can click the context menu at the upper-left side of the **SAP Logon** window, and then click **Close** to close the application.

Problem	Description	Solution
On the SAP server screens, the playback fails for some of the controls.	During the recording, even though the controls are captured correctly, when you use the **Highlight** option from the Object map, the correct control might not be highlighted. These controls might not be recognized correctly during the playback and the playback fails.	You can do the following steps: 1. Insert controls by using the Insert test object tool. 2. Update the existing control name with the inserted control name.
The recording monitor does not display the steps as soon as they are recorded.	When you record the test, the recording monitor is updated with each step that you record. But, sometimes the steps are not displayed in the recording monitor.	Due to buffering, the test steps might be updated in the recording monitor with some delay. You can proceed with recording. The steps are displayed in the recording monitor after the buffering is completed.

Problems with environment enablement

When it is not possible to record on the application-under-test, you must verify whether the test environment was enabled for functional testing.

Problem

Rational® Functional TesterHCL OneTest™ UI is unable to record on the application-under-test, or it records incorrect statements.

Resolution

Verify that the domain was enabled for functional testing. For example, for HTML applications, verify whether the browsers and their associated Java Runtime Environments (JREs) were enabled for testing. For information about preparing the functional test environment for testing, see Preparing the functional test environment on page 774.

In Rational® Functional TesterHCL OneTest™ UI version 8.2.2 version 9.1 and later, the test environment is automatically enabled for functional testing under certain conditions; no manual enablement is required. For information about the conditions in which automatic enablement occurs, see Automatically enabled environment for functional testing on page 774.

If the environment is not automatically enabled, you must enable the components manually. For information about enabling components manually, see these topics:

- Enabling web browsers on page 780
- Enabling Java environments on page 778

Handling exceptions

If an exception occurs during testing, you might be able to handle it by following these instructions. Two common exceptions are the ambiguous recognition exception and the Mutex timeout exception during playback.

- Ambiguous recognition exception on page 1558
- Mutex timeout exception during playback on page 1558

Ambiguous recognition exception

Problem

In certain cases during playback, for example, when multiple instances of a browser are running, Rational® Functional TesterHCL OneTest™ UI might be unable to differentiate between two similar objects in the software that is being tested. At such times, an AmbiguousRecognition exception occurs. For example, this problem might occur when multiple instances of a browser are running.

Resolution

1. Find the duplicate instance of the application, close the instance, and click **Retry**.
2. Add a unique property to the object in the test object map to distinguish the ambiguous objects.
3. Open the test object map from the Script Explorer.
4. Find the object that the ambiguous exception was thrown for, right-click the object, and highlight it. This action highlights the objects that Rational® Functional TesterHCL OneTest™ UI finds similar.
5. Identify a unique property among these objects and use the `find()` API, passing the unique property to find the method, and then do the required operation. For example, to click a button that is named Back, use this approach:

```
TestObject [] backBtn = find(atDescendant(".class", ".PushButton", ".text", "Back"));
if( backBtn.length == 1){
 ((GuiTestObject)backBtn[0]).click();
}
else{
 //Add code to log message that more than one instance of object is still found
}
```

Mutex timeout exception during playback

Problem

During the playback of functional test scripts, this Mutex timeout exception might occur:`com.rational.test.ft.svs.Mutex$TimeoutException` .

Resolution

The default SpyHeapSize that is set in Rational® Functional TesterHCL OneTest™ UI is 1048576. Increase this value by adding a DWORD value named SpyHeapSize to the registry and setting its value to 2097152. Complete these steps:

1. Click **Start** and then click **Run**. Type regedit. The Windows Registry Editor opens.
2. Navigate to the `[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rational Test\8]` `[HKEY_LOCAL_MACHINE\Software\HCL Technologies\HCL OneTest UI` registry key.
3. Create a new DWORD value named SpyHeapSize and set its value data to 2097152 (or 200000 hexadecimal)

You can also add the SpyHeapSize key by creating and then running a `.reg` file that has these contents:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rational Test\8]
"SpyHeapSize"=dword:00200000
```

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\Software\HCL Technologies\HCL OneTest UI]
"SpyHeapSize"=dword:00200000
```

Collecting Rational® Functional TesterHCL OneTest™ UI error logs

While working with Rational® Functional TesterHCL OneTest™ UI, you might encounter some problems that you can easily troubleshoot. If you are unable to troubleshoot the problem, you can contact IBM Software Support. Gather all the required background information such as logs, crash dumps, and traces and provide them to the IBM Software Support for investigation. In addition to the background information, you must also generate the error logs and provide those files to the support team.

About this task

To generate Rational® Functional TesterHCL OneTest™ UI error logs:

1. Set the preferences for logging and tracing in the Logging and Tracing page. Click **Window > Preferences**, expand **Functional Test** in the left pane, and then click **Logging and Tracing**. You can collect the errors, warning, and information messages into a log file (rft_log.txt). The debug information is collected as trace date into the trace file. The trace file can be either a .txt file or a .log file, depending on your specification in the Logging and Tracing page. Trace files in the .log format can be imported into the Eclipse Error Log view within Rational® Functional TesterHCL OneTest™ UI for viewing and filtering. You can specify the log file and the trace file directory.
2. Perform the operation that caused the problem.

3. You can send the generated rft_log.txt and rft_trace.txt or rft_trace.log files to IBM Software Support.

Note: You can identify the files by looking at the timestamp of the generated files.

Viewing trace files within Rational® Functional TesterHCL OneTest™ UI

You can view Rational® Functional TesterHCL OneTest™ UI trace files in the Rational® Functional TesterHCL OneTest™ UI integrated development environment (IDE) Error Log view.

Before you begin

Complete these steps:

1. Set the preferences for trace files in the Logging and Tracing page. Click **Window > Preferences**, expand **Functional Test** in the left pane, and then click **Logging and Tracing**.
2. Enable the generation of trace files by selecting the **Enable Tracing** check box
3. Enable the generation of trace files in the Eclipse error log format (.log) by selecting the **Generate traces in Eclipse error log format** check box.
4. Specify other details for the trace files, such as the level of detail, allowable file size, number of files to be retained, and the location where the trace files should be stored, and save your settings.

About this task

Rational® Functional TesterHCL OneTest™ UI trace files contain debug information which you can use to troubleshoot problems you encounter. You can configure Rational® Functional TesterHCL OneTest™ UI to generate trace files in the .txt format, or the Eclipse error log (.log) format. Only trace files generated in the .log format can be imported into the Eclipse Error Log view within Rational® Functional TesterHCL OneTest™ UI. After importing the .log trace file into the Error Log view, you can work with the data using Eclipse log filtering operations.

1. In Rational® Functional TesterHCL OneTest™ UI, click **Window > Show View**, and then click **Other**.
2. In the **Show View** dialog box, expand **General**, and click **Error Log**.

   Result

   The Error Log view is displayed as a tab in Rational® Functional TesterHCL OneTest™ UI.
3. In the Error Log view, click the Import Log icon.
4. Navigate to the directory where the trace files are stored, and select the trace file (rft_trace.log) to import.

   Result

   The trace file details are shown in the Error Log view.
5. You can use the Eclipse filter operations to work with the trace data. Right-click a trace data item to open the **Log Filters** dialog box. You can filter event types, limit the number of visible items in the log, filter events from the most recent session, and enable filters to hide stack trace elements.

AutoPD Collection artifacts in IBM® Support Assistant

IBM® Rational® Functional TesterHCL OneTest™ UI, contributes a few files that the Automated Problem Determination (AutoPD) Collector tool in the IBM® Support Assistant application uses.

The following files from the functional testing product are contributed to the AutoPD Collector tool:

- The user preferences file, and the logging and tracing preferences file.
- The extension preferences file.
- A collection of browser details from Internet Explorer (only on Microsoft® Windows®) and Firefox (on both Microsoft® Windows® and Linux®)
- A collection of Microsoft® .NET framework details (only on Microsoft® Windows®)

Frequently asked questions

This section provides answers to frequently asked questions about IBM® Rational® Functional TesterHCL OneTest™ UI.

Frequently asked questions about Rational® Functional TesterHCL OneTest™ UI

For answers to some generic questions on using IBM® Rational® Functional TesterHCL OneTest™ UI, see this topic.

- Does Rational Functional TesterHCL OneTest UI support the testing of my application? on page 1561
- Can Rational Functional TesterHCL OneTest UI be used to test Eclipse-based applications? on page 1562
- Is the Eclipse integrated development environment (IDE) provided with Rational Functional TesterHCL OneTest UI? on page 1562
- How do I enable debugging in Rational Functional TesterHCL OneTest UI? on page 1562
- How do I transfer the information specified in the Rational Functional TesterHCL OneTest UI Application Configuration Tool to playback agent machines? on page 1562
- How do I run Rational Functional TesterHCL OneTest UI under a different Java™ Runtime Environment (JRE)? on page 1563
- How do I cleanly uninstall Rational Functional TesterHCL OneTest UI versions 7.x, 8.0 and 8.1.x on Microsoft Windows? on page 1564
- How do I enable the browser environments for testing applications on Microsoft Windows systems compliant with Federal Desktop Core Configuration (FDCC)? on page 1564
- Can Rational Functional TesterHCL OneTest UI be used with a project enabled for Unified Configuration Management (UCM)? on page 1564
- Does configuring the application under test (AUT) modify the AUT? on page 1564
- What happens when Internet Explorer is enabled? on page 1565
- What are the language limitations for Rational Functional TesterHCL OneTest UI scripts? on page 1565
- How to use a customized DLL in Rational Functional TesterHCL OneTest UI for Microsoft® .NET? on page 1565

Does Rational® Functional TesterHCL OneTest™ UI support the testing of my application?

Rational® Functional TesterHCL OneTest™ UI supports applications that are developed using certain technologies. Verify the type of technology that is used to develop the test application, and verify if Rational® Functional TesterHCL OneTest™ UI supports functional testing of the domains and the controls in the application. If Rational® Functional

TesterHCL OneTest™ UI supports the technology, and does not support a specific control for functional testing by default, you can use Proxy SDK to develop proxies that enable support for specific controls.

For information about supported domains, see Test application domain support on page 1703.

For information about using Proxy SDK, see Introduction to Proxy SDK on page 1078.

Can Rational® Functional TesterHCL OneTest™ UI be used to test Eclipse-based applications?

Yes. For configuration details, see Eclipse Support on page 1711.

Is the Eclipse integrated development environment (IDE) provided with Rational® Functional TesterHCL OneTest™ UI?

If your Rational® Functional TesterHCL OneTest™ UI installation detects another Rational Software Development Platform (SDP) tool HCL product on the workstation or an existing Eclipse installation, it shares the IDE shell on the workstation. If no other SDP HCL tool is found on the workstation, Rational® Functional TesterHCL OneTest™ UI installs its own instance of the shell. Rational® Functional TesterHCL OneTest™ UI shares the shell with Rational® Software Architect, Rational® Application Developer, Rational® Performance TesterHCL OneTest™ Performance, or any of the Rational Software Development Platform offerings HCL products.

For information about shell sharing, see the technote http://www.ibm.com/support/docview.wss?&uid=swg27038243.

How do I enable debugging in Rational® Functional TesterHCL OneTest™ UI?

When you encounter a problem, debugging might be helpful to obtain more information about the possible causes of the problem. The problem can be in playback as well as in recording. To set the Rational® Functional TesterHCL OneTest™ UI Debug Perspective preference, see the instructions in Debugging scripts on page 1449.

How do I transfer the information specified in the Rational® Functional TesterHCL OneTest™ UI Application Configuration Tool to playback agent machines?

Rational® Functional TesterHCL OneTest™ UI scripts contain startApp API calls to start your application under test at run time. For example:

```
startApp("IBM.com");
```

```
startApp("HCL.com");
```

At run time, the startApp command matches the string that is passed to it, with the corresponding entry in the Application Configuration Tool on the local playback machine. If no matching entry is found, an exception occurs and playback fails:

```
com.rational.test.ft.script.RunException: CRFCN0630E: Cannot find application [IBM] in the configuration
 file.
```

```
com.rational.test.ft.script.RunException: CRFCN0630E: Cannot find application [HCL] in the configuration
 file.
```

To avoid this exception, applications must be configured for testing on the playback workstation before the scripts are played back. For instructions to configure your applications for testing, see Configuring applications for testing on page 796.

To transfer your configured applications list between workstations, do these steps:

1. Open the Application Configuration Tool (click **Configure > Configure Applications for Testing**).

   Alternately (or if you do not have a scripting environment installed), use the command line interface as described in the technote Invoking the Functional Test Configure menu items from a command line. If you have multiple applications to test, manually specifying the application information in the Application Configuration Tool can be time consuming. This technote describes how to copy the application configuration information between workstations, avoiding the need to manually enter this information.
2. Open the `C:\ProgramData\IBM\RFT\configuration\configurations.rftcfg` `C:\ProgramData\HCL\OneTestUI\configuration\configurations.rftcfg` file in a text editor. This XML file stores the information specified in the Application Configuration Tool.
3. Copy the information contained within these tags:

   ```
   <ApplicationList L=".ApplicationList">
   </ApplicationList>
   ```
4. Paste this information in the `C:\ProgramData\IBM\RFT\configuration\configurations.rftcfg` `C:\ProgramData\HCL\OneTestUI\configuration\configurations.rftcfg` file on your additional workstations.
5. Save the changes and restart Rational® Functional TesterHCL OneTest™ UI. The updated information is displayed in the Application Configuration Tool.

How do I run Rational® Functional TesterHCL OneTest™ UI under a different Java™ Runtime Environment (JRE)?

In a normal Eclipse-based Rational® Functional TesterHCL OneTest™ UI installation, Rational® Functional TesterHCL OneTest™ UI uses its own JRE, or the JRE of the Eclipse-based shell in which it has been installed. However, you can specify the JRE to be used, if required. Use the `-vm` argument at the command prompt to specify the JRE to be used:

```
"C:\Program Files\IBM\SDP\eclipse.exe" -vm "C:\Progra~1\j2sdk1.4.1_02\bin\javaw.exe"
```

```
"C:\Program Files\HCL\HCLOneTest\eclipse.exe" -vm "C:\Progra~1\j2sdk1.4.1_02\bin\javaw.exe"
```

Note: The Java version is only provided as an example. You can specify any Java version that is supported by Rational® Functional TesterHCL OneTest™ UI.

You can also change the JRE permanently by editing the `eclipse.ini` file in the SDP directory or by editing this registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rational Test\8\Rational FT JRE
```

```
HKEY_LOCAL_MACHINE\SOFTWARE\HCL Technology\HCL OneTest UI\HCL FT JRE
```

Note: These steps to change the JRE are applicable only to Java-based Rational® Functional TesterHCL OneTest™ UI installations and not to installations in Microsoft Visual Studio for .NET.

How do I cleanly uninstall Rational® Functional TesterHCL OneTest™ UI versions 7.x, 8.0 and 8.1.x on Microsoft® Windows®?

See Uninstalling Rational Functional TesterHCL OneTest UI cleanly on page 1552.

How do I enable the browser environments for testing applications on Microsoft® Windows® systems compliant with Federal Desktop Core Configuration (FDCC)?

Do these steps if you encounter problems trying to enable the browser environment for testing applications on FDCC compliant Microsoft® Windows® computers:

To enable the Internet Explorer 7 and Internet Explorer 8 browsers:

Note: This workaround is not required when you enable these browsers in Rational® Functional TesterHCL OneTest™ UI, version 8.1.1.2.

1. Click **Start > Run**. In the **Run** window, type `regedit`.
2. In the `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Internet Explorer\Main\Enable Browser Extensions` key in the Registry Editor, enable the third party browser extensions.

Note: Ensure that the Java Runtime Environment (JRE) corresponding to Internet Explorer 7 and Internet Explorer 8 is properly enabled and aligned with the browser.

To enable the Mozilla Firefox browser:

1. Click **Tools > Add-Ons**.
2. Navigate to the `<product installation directory>\FunctionalTester\bin\enabler` directory.
3. Drag the enabler files `RtxFFEnabler.xpi` (for Firefox 2.0) , `RtxFF3Enabler.xpi` (for Firefox 3.0) , `RtxFF35Enabler.xpi` (for Firefox 3.5) and `RtxFF36Enabler.xpi` (for FireFox 3.6) to **Add-Ons** in Firefox.
4. Click **Install**.

Can Rational® Functional TesterHCL OneTest™ UI be used with a project enabled for Unified Configuration Management (UCM)?

Yes. However, Rational® Functional TesterHCL OneTest™ UI supports only single-stream UCM.

Does configuring the application under test (AUT) modify the AUT?

Configuring an application does not modify it. It is analogous to setting up a system of shortcuts so that Rational® Functional TesterHCL OneTest™ UI can start the application.

What happens when Internet Explorer is enabled?

Internet Explorer is enabled by registering a browser helper object (BHO) with Internet Explorer. In the registry, this key is added if Internet Explorer has been properly enabled:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper
 Objects\{1E9FB1C4-F40B-4E10-898E-D6209B122F6B} Rational IE Enabler
```

What are the language limitations for Rational® Functional TesterHCL OneTest™ UI scripts?

Rational® Functional TesterHCL OneTest™ UI uses standard Java. All the normal Java functions are available in functional test scripts and no subset of the language with custom extensions is used.

How to use a customized DLL in Rational® Functional TesterHCL OneTest™ UI for Microsoft® .NET?

Perform one of the following actions to add a customized DLL:

- Add the DLL file to the Rational® Functional TesterHCL OneTest™ UI `<install_dir>\bin` directory, and then add the name of the DLL file to `netAssembliesForCompiler.txt` in the same directory.
- Add the DLL file to the `C:\ProgramData\IBM\RFT\customizationC:\ProgramData\HCL\HOTUI\customization` directory.

Frequently asked questions about recording scripts with Rational® Functional TesterHCL OneTest™ UI

This topic provides answers to some frequently asked questions about recording functional test scripts using IBM® Rational® Functional TesterHCL OneTest™ UI.

- Which controls does Rational Functional TesterHCL OneTest UI support for functional testing? on page 1565
- Is there only one dataset for a script? on page 1566
- How do I test a popup window with Rational Functional TesterHCL OneTest UI? on page 1566
- Is it required to start an application with Rational Functional TesterHCL OneTest UI to make it testable? on page 1566
- Can scripts recorded under a JRE be played back on a different JRE? on page 1567
- Does Rational Functional TesterHCL OneTest UI support recording against the menu in browsers? on page 1567

Which controls does Rational® Functional TesterHCL OneTest™ UI support for functional testing?

For a list of supported controls, see the following topics:

- Adobe PDF documents support on page 1704
- Dojo support on page 1709
- Flex support on page 1711
- .NET support on page 1717

- Windows support on page 1720
- PowerBuilder support on page 1722
- SAP support on page 1724
- Visual Basic support on page 1725

See the technical document at http://www.ibm.com/support/docview.wss?&uid=swg27038243 for a list of domains supported for functional testing in each release of Rational® Functional TesterHCL OneTest™ UI.

Is there only one dataset for a script?

Yes. datasets can be private or shared, much like object maps, so several scripts may use the same dataset.

In Simplified Scripting, more than one shared dataset can be used at a group level but not at a script level.

How do I test a popup window with Rational® Functional TesterHCL OneTest™ UI?

Rational® Functional TesterHCL OneTest™ UI has a special function to test popup windows. Do these steps to capture a popup window on the application under test (AUT):

1. Begin recording and open the AUT.
2. Click the **Insert Verification Point or Action Command** button  on the Recording Monitor toolbar.
3. Select **Time Delayed Selection** from the **Selection Method** list. Specify a time to delay before selection to allow sufficient time to navigate to the popup window and display it; for example, 10 seconds.
4. Click the object finder.

   As the timer counts down, move the mouse into the popup window, without actually pointing to a specific object. A good place to move the mouse is between the items on the list. Make sure that the mouse pointer is a regular pointer, not a pointing hand, and that you are not pointing to a specific item in the menu, unless that is all you want to capture. When the time expires, Rational® Functional TesterHCL OneTest™ UI highlights the data that will be captured in red and returns to the Verification Point dialog box. Here, you can choose to do a data verification point and test the table contents.

   For information about the Time Delayed Selection object selection method, see Select an Object page of the Verification Point and Action Wizard on page 1827.
5. After clicking **Next**, you see the popup window content.

Is it required to start an application with Rational® Functional TesterHCL OneTest™ UI to make it testable?

No. If the environment has been correctly enabled, the application is testable even if Rational® Functional TesterHCL OneTest™ UI is not used to start it.

For information about enabling environments, see Preparing the functional test environment on page 774.

Can scripts recorded under a JRE be played back on a different JRE?

Scripts recorded under a JRE can be played back on a different JRE without requiring maintenance. However, when recording on JRE 1.3.x and playing back under JRE 1.4.x, object properties might change due to changes in the JRE. This causes errors during playback. To eliminate the errors, you must update the object properties.

For information about updating object properties, see Updating recognition properties.

Does Rational® Functional TesterHCL OneTest™ UI support recording against the menu in browsers?

Rational® Functional TesterHCL OneTest™ UI does not support recording against the menu in Internet Explorer or Firefox browsers. Only toolbar buttons that are common to both Internet Explorer and Firefox are supported: Back Bookmarks (Firefox) and Favorites (Internet Explorer), Close, Forward, Home, Maximize, Minimize, Search (push button only), Stop. The HTML support of Rational® Functional TesterHCL OneTest™ UI is targeted at cross-browser support, and attempting to support the menu in the two targeted browsers, Internet Explorer and Firefox, would lead to incompatibilities.

See the topic Tips and tricks for functional testing HTML applications on page 1543 in the information center.

Related information

Working with functional test scripts (Windows-only) on page 865

Frequently asked questions about object recognition and object maps in Rational® Functional TesterHCL OneTest™ UI

This topic provides answers to some frequently asked questions about object recognition and test object maps.

- What is an object map and why is it needed? on page 1568
- What happens if two different objects have the same name? on page 1568
- What is the difference between a private and public object map? on page 1568
- What do the weights on the recognition properties mean and how are they used? on page 1568
- Can the default weights that are assigned to each property be modified? on page 1568
- Can a property weight or value be modified for all objects in a project or all objects of a specific type? on page 1568
- Can scripts that were created with private maps be merged to a public map? on page 1569
- Can a private map be made public? on page 1569
- Do changes to the object map hierarchy need new sections in the object map for all child objects? on page 1569
- Can properties that are used for object recognition be modified? on page 1569
- Can a script reference more than one object map? on page 1569
- Can individual objects be copied from one map to another? on page 1569
- What are the performance metrics of an object map as it grows in size? on page 1570

What is an object map and why is it needed?

An object map contains each object that was acted upon during recording, as well as the object's recognition properties. For each property, the map also contains a weight that indicates how heavily Rational® Functional TesterHCL OneTest™ UI relies on that property for recognition.

The object map can be automatically populated when a script is recorded, or manually by adding objects to the map.

The object map aids efficient script maintenance. When objects in the application under test are modified, the object map is a single source that can be updated. By updating the map, all scripts that reference the modified object use the updated object information.

What happens if two different objects have the same name?

Rational® Functional TesterHCL OneTest™ UI differentiates objects based on recognition properties and hierarchy. For example, if a button exists in a different top-level window, then it appears as a different button in the map. If two or more different objects have the same name, to make the name unique, Rational® Functional TesterHCL OneTest™ UI adds number suffixes to the name of the object that is second, third, and so on.

If there are two identical instances of any objects, such as two instances of a browser, Rational® Functional TesterHCL OneTest™ UI provides a method to handle identification.

What is the difference between a private and public object map?

A private object map is used by a single script. A public (shared) object map is used by multiple scripts. The default setting is for each script to use a private object map. This is appropriate for single users who are starting to learn the tool. However, a team of testers working on a common application should use a shared object map so that they can take advantage of objects being globally defined in a single map.

What do the weights on the recognition properties mean and how are they used?

The weights indicate how important a specific property is for identifying an object during script execution. The possible settings are 0 (not important) to 100 (very important). You can change the weights of most of the properties. The `.class` property is fixed and cannot be changed.

If you set the weight to 0, the value of that property is ignored when attempting to identify an object.

Can the default weights that are assigned to each property be modified?

Use the Object Property Configuration Tool to assign new default weights to properties and also define new properties to be used when identifying objects in future recordings.

Can a property weight or value be modified for all objects in a project or all objects of a specific type?

You can use the Object Map Find and Modify utility to find all objects that match a criteria such as property name, property value, or various custom filters. Actions that can be taken on the matching objects include Add Property,

Remove Property, Change Value, and Change Weight. Modifications can be applied to objects either one at a time or globally.

Can scripts that were created with private maps be merged to a public map?

Yes. To merge the scripts, create a new public map by clicking **File > New > Test Object Map**. If a wizard page opens, close it. Right-click the created map and select **Merge Objects into**. In the dialog box that is displayed, select the scripts to merge. Ensure that you select the **Connect selected Functional Test scripts** check box to attach the selected scripts to the new merged object map.

For information about merging multiple maps, see Merging multiple test object maps.

Can a private map be made public?

Yes. The contents of a private map can be copied to a public map by doing these steps:

1. Select **File > New > Test Object Map**.
2. Select the folder where you want the new public map created and then type the new map name. You can also select the check box to set this test object map as the default choice for new Rational® Functional TesterHCL OneTest™ UI scripts. Click **Next**.
3. Select **Test Object Maps and scripts to copy Test Objects from**.
4. Select the script that contains the private map that you want to be made public. You can also select the **Connect selected scripts with the new Test Object Map** check box.

Do changes to the object map hierarchy need new sections in the object map for all child objects?

If the properties of parent node change due to insertions or deletions, or a new node is added to the hierarchy, you can update the map. To update the map, use the map editor to insert references to objects that have a newly inserted parent. You then have duplicate references for the child objects of the new object. For each of these references, drag the old object to the new, so that the map editor unifies the objects. You can use the same steps to delete an object from the map hierarchy.

Can properties that are used for object recognition be modified?

Yes. Open the object in the object map then select **Test Object > Update Recognition properties**. Use the object finder to point to the object. When the Update Recognition Properties dialog box with the three panes appears, right-click a property in the All Active Properties pane and select **Add to Unified Test Objects Properties**.

Can a script reference more than one object map?

No.

Can individual objects be copied from one map to another?

No.

What are the performance metrics of an object map as it grows in size?

As an object map grows, the time required to open and load the map increases.

When a script runs, the object map is loaded the first time it is needed during the run. If you have a test case suite that uses functional test scripts, the map loads as each script runs, because each script is a separate process. A typical application would have approximately 2000-3000 objects (some maps could be larger or smaller).

The metrics shown in Table 1 were computed at script run time but can also apply when the map is loaded. They show that a significant increase in map load time does not occur till a map well beyond normal size is encountered. Playback time increases only at the beginning of the script run when the map is loaded, and it does not slow down the script run after the map is loaded. Additionally, nested scripts (using the callScript function) that share the same object map also share the same map instance during script run. Hence, nested scripts do not increase the load time.

Table 56. Object map metrics

Objects in map	Seconds for total script execution
10	X seconds
600 - 2000	X + 2 seconds
2000 - 11000	X + 3 seconds
more than 11000	X +10 seconds

Related information

Working with functional test object maps in the Visual Studio IDE on page 903

Frequently asked questions about integrations with Rational® Functional TesterHCL OneTest™ UI

This topic provides answers to some frequently asked questions about IBM® Rational® Functional TesterHCL OneTest™ UI integrations with other Rational products.

- Does Rational Functional TesterHCL OneTest UI integrate with test management systems? on page 1571
- What are the benefits of connecting a functional test project to a test management solution? on page 1571
- How does keyword testing work ? on page 1571
- When working with keywords, when are licenses for Rational Quality Manager and Rational Functional TesterHCL OneTest UI required? on page 1571
- Can Rational Functional TesterHCL OneTest UI be installed with Rational Application Developer or Rational Software Architect? on page 1571
- Which source control management tools does Rational Functional TesterHCL OneTest UI integrate with? on page 1572

- Why should I use source control management systems with Rational Functional TesterHCL OneTest UI? on page 1572
- Can Rational Functional TesterHCL OneTest UI be used without a source control system? on page 1572

For information about the compatible versions of Rational products that can be integrated with Rational® Functional TesterHCL OneTest™ UI, see the technical documents athttp://www.ibm.com/support/docview.wss?&uid=swg27036168 and http://www.ibm.com/support/docview.wss?&uid=swg27036169.

Does Rational® Functional TesterHCL OneTest™ UI integrate with test management systems?

Rational® Functional TesterHCL OneTest™ UI can be integrated with Rational® Quality Manager.

What are the benefits of connecting a functional test project to a test management solution?

Rational® Functional TesterHCL OneTest™ UI is a test implementation application. On its own, it does not provide capabilities for test planning, test design, sophisticated test execution options, or in-depth results analysis. These capabilities come from a test management system that can use Rational® Functional TesterHCL OneTest™ UI as one of several possible implementation and automation tools.

How does keyword testing work ?

Rational® Quality Manager users define the manual tests in the Rational® Quality Manager editor. They can identify test steps or sets of steps as keywords. These keywords are logical groupings of steps that can be reused across multiple manual tests.

The keywords are also visible within Rational® Functional TesterHCL OneTest™ UI. An automation specialist can select a keyword and can record or associate it with a Rational® Functional TesterHCL OneTest™ UI automated test. When the test is run, the keyword can be executed as an automated test. This provides limited automation specialists with the ability to automate the most high leverage test steps.

When working with keywords, when are licenses for Rational® Quality Manager and Rational® Functional TesterHCL OneTest™ UI required?

To create manual tests, define keywords, and run tests containing keywords implemented with manual steps, a Rational® Quality Manager license is needed. To record automated tests as implementations of keywords, Rational® Functional TesterHCL OneTest™ UI bits and license are needed. To run a manual test with automated keywords, both Rational® Quality Manager and Rational® Functional TesterHCL OneTest™ UI must be installed, as well as a Rational® Functional TesterHCL OneTest™ UI license.

Can Rational® Functional TesterHCL OneTest™ UI be installed with Rational® Application Developer or Rational® Software Architect?

Rational® Functional TesterHCL OneTest™ UI installs as a perspective into Rational® Application Developer or Rational® Software Architect.

For information about shell sharing with other Rational products, see the technical document at http://www.ibm.com/support/docview.wss?&uid=swg27036168.

Which source control management tools does Rational® Functional TesterHCL OneTest™ UI integrate with?

Rational® Functional TesterHCL OneTest™ UI can be integrated with IBM® Rational® Team Concert™.

Why should I use source control management systems with Rational® Functional TesterHCL OneTest™ UI?

Rational® Functional TesterHCL OneTest™ UI can integrate with source control management systems such as Rational® Team Concert™ to manage concurrent changes to test assets and to version test scripts changes. The Eclipse Shell (WSW), which iRational® Functional TesterHCL OneTest™ UIs the integrated development environment (IDE), assumes the use of a version control system. So, in a team environment where you want to share scripts, script templates, or object maps with others in your testing team, and prevent others from overwriting test assets, a source control management system is useful. Source control management systems also provide the benefit of storing versions of test scripts, merging scripts, and allowing branching.

For information about shell sharing with other Rational products, see the technical document at http://www.ibm.com/support/docview.wss?&uid=swg27038243.

Can Rational® Functional TesterHCL OneTest™ UI be used without a source control system?

Yes, provided the environment is either a single tester working in a datastore, or testers who are not sharing any object maps or scripts. Rational® Functional TesterHCL OneTest™ UI can also be used without a source control system if the environment is a team of testers sharing assets in a datastore, but you might still need some specific procedures to prevent users from overwriting each other's work.

Rational® Functional TesterHCL OneTest™ UI error messages

This section provides information about error messages that you might encounter with Rational® Functional TesterHCL OneTest™ UI. This section lists the error messages by ID, explanation, system action and your response to correct the error message.

CRFCC0006E

Object Maps not merged because target map file is read-only

Explanation: The target map file cannot be changed because the file is read-only.

User response: Check the permissions of the target map file. If necessary, change the read-only status of the file, and try to merge the maps again.

CRFCC0007E

Map merge problems

Explanation: An internal error occurred during the object map merge operation.

System action: The merge operation fails

User response: Try to merge the object maps again. If the problem persists contact support

CRFCC0008E

Script Definition: Merge: The test object name is a reserved word *test_object*. Rename the test object {0}, and then check the script in.

Explanation: The *test_object* test object name is a reserved word and cannot be used as a name.

User response: Rename the *test_object* test object, and check in the script.

CRFCC0009E

Option *option_name* is not defined

Explanation: The option definition is null. The option must be defined. The option definitions are stored in the rational.rftcust file in `FunctionalTester\bin` folder

User response: Check the defined options, and specify only a defined option. The option definitions are stored in the rational.rftcust file in the `FunctionalTester\bin` folder.

CRFCC0010E

-*option_name* option must be preceded by a -from mapFileName option.

Explanation: A -from mapFileName option and source file did not precede the *option_name* option.

System action: Execution of the command fails.

User response: Specify the -from mapFileName option and source file before the option, and try the operation again.

CRFCC0011E

-*option_name* option must be preceded by a -to mapFileName option.

Explanation: A -to mapFileName option and source file did not precede the *option_name* option. The correct option and source file name must be precede the *option_name* option.

User response: Specify the -to mapFileName option and source file before the option and try the operation again.

CRFCC0012E

The -*option_name* option must be followed by a map file name.

Explanation: The *option_name* option must be followed by a map file name.

User response: Specify the map file name after the option and try the operation again.

CRFCC0013E

Unable to complete the operation. The file *file_name* could not be copied. Typically, this is because you are out if disk space.

User response: Check disk space. If enough space is available, check error message detail for information about a resolution, and try to copy the file again.

CRFCC0014E

Unable to complete the operation. The file *file_name* could not be copied. Typically, this is because you are out if disk space.

User response: Check disk space. If enough space is available, check the error message detail for information about a resolution, and try to copy the file again.

CRFCC0015E

Unable to create file *file_name*. Check to see if you have sufficient disk space.

Explanation: The target disk might not have sufficient space. The *file_name* file could not be created; there is no backup of the file to be merged.

User response: Check the disk space. If enough space is available, check the error message detail for information about a resolution, and try to merge the files again.

CRFCC0018E

Unable to merge the directory. You must remove the conflict by renaming the file or script, or complete the graphical merge.

Explanation: A conflict with the name of the file or the script is preventing the merge. The script and file names must be unique.

User response: Remove the conflict by renaming the file or script, and try to merge the files again.

CRFCC0020E

Unable to copy *file_name* to *file_path*. Copy the file manually.

User response: Copy the file manually.

CRFCC0021E

Unable to checkout *file_path*. Make sure you have enough disk space.

Explanation: Disk space limitations might prevent making changes to the cached version of the file to reflect changes made to the state such as checked in, checked out, or hijacked.

System action: Changes that are made to the state are not saved.

User response: Check disk space and ensure that sufficient space is available; then try the operation again.

CRFCC0022E

Unable to copy *current_path* to *destination_path*

User response: Check the log file for information related the copying the directory.

CRFCC0023E

Unable to rename the file *file_path*

User response: Check the log file for information related to renaming the file.

CRFCC0024E

Unable to checkout *file_path*

Explanation: There might be a write-protected file with the same name in the destination folder.

System action: No changes to the cache are made to reflect changes in the state.

User response: Check the log file for information related the checking out the file.

CRFCC0030W

You cancelled the merge. The selected item is not merged or checked in.

System action: The merge was canceled.

User response: You can perform the merge and checkin later if necessary.

CRFCC0031E

You cancelled the merge. The shared map was merged and checked in. None of the other files in the script were checked in.

System action: The merge was canceled.

User response: Make the changes to the files and perform the merge operation and check in the files. If you do not want to perform the merge operation, check in the files with or without any changes.

CRFCC0032E

You cancelled the merge. None of the files in the script were checked in.

System action: The merge was canceled.

User response: Merge the files before checking in the files.

CRFCC0033E

The checkin did not complete because object map could not be merged.

Explanation: The object map is not merged and a later version of the map file exists when you try to check in the script

System action: The check in fails because the merge operation is not completed

User response: Merge the object maps, and then try to check in the script.

CRFCC0034E

Unable to update the Script Helper file. Select **Script > Update Script Helper** to re-create the helper. Then check in the script.

User response: Click **Script > Update Script Helper** to create the helper again. Check in the script after updating the Script Helper.

CRFCC0035E

Selected project is not a Functional Test project. Specify another repository type.

System action: The project is not loaded.

User response: Specify a functional test project to load.

CRFCC0037E

Unable to perform operation. The functional test script needs to be checked-out before performing this operation.

Explanation: The functional test script is not checked out. The script must be checked out before performing this operation. The script is not storing the script definition to persistent storage.

User response: Check out the test script, and try the operation again.

CRFCN0001E

An invalid subitem was specified or Rational® Functional TesterHCL OneTest™ UI does not support the specified subitem.

Explanation: Rational® Functional TesterHCL OneTest™ UI requires valid subitems that the application supports.

System action: Recording of the subitem stops.

User response: Verify that the subitem is valid and that Rational® Functional TesterHCL OneTest™ UI supports the subitem, and try recording again.

CRFCN0002E

Attempt to perform mouse operation on co-ordinates that are off-screen.

Explanation: Two situations cause this error. When a screen snapshot has to be taken during playback and the control is not fully visible in the screen, the mouse action cannot be completed. When a point that is off the screen is clicked, the mouse action cannot be processed.

System action: Playback stops and the message is displayed in the Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Ensure that the control is not off the screen. The control has to be completely visible within the desktop screen coordinates.

CRFCN0003E

Attempt to perform mouse operation on co-ordinates that are off-screen

Explanation: Two situations cause this error. When a screen snapshot has to be taken during playback and the control is not fully visible in the screen, the mouse action cannot be completed. When a point that is off the screen is clicked, the mouse action cannot be processed.

System action: Playback stops and the message is displayed in the Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Ensure that the control is not off the screen. The control has to be completely visible within the desktop screen coordinates.

CRFCN0004E

The requested action cannot be completed

Explanation: The object in the application under test might not support the action or Rational® Functional TesterHCL OneTest™ UI might not support the action. For example, this error might occur if a click action cannot be performed on an object in the application under test.

System action: Recording or playback fails.

User response: Verify that Rational® Functional TesterHCL OneTest™ UI supports testing the specified object and that the application object supports the action that you are attempting.

CRFCN0005E

Operation not supported on UNIX

Explanation: This operation is not supported on Unix environment.

System action: The operation stops.

User response: Use an alternative Unix function to perform the same operation.

CRFCN0006E

Operation not supported on UNIX

Explanation: The specified operation is not supported on Unix environment.

System action: The operation stops.

User response: Use an alternative Unix function to perform the operation.

CRFCN0007E

The action cannot be completed.

Explanation: Action failed during the retry operation after waiting for the specified time. Retrying the action failed even after waiting for the specified time. Testing the object must be supported by Rational® Functional TesterHCL OneTest™ UI and the object being tested must support the attempted action.

System action: Operation stops.

User response: Verify that Rational® Functional TesterHCL OneTest™ UI supports testing the specified object and that the object supports the attempted action.

CRFCN0008E

Attempt to use a registered object after its associated object has been disposed.

Explanation: This error message is displayed when Rational® Functional TesterHCL OneTest™ UI internally accesses a native web element reference that is not valid.

System action: This message does not cause any problem in playing back the script. This message is displayed on Rational® Functional TesterHCL OneTest™ UI console. Rational® Functional TesterHCL OneTest™ UI automatically handles the error by getting the valid reference.

User response: Contact support in case of playback failure with this message.

CRFCN0009E

An attempt was made to use a {0} type TestObject after its associated object had been disposed of.

Explanation: The application internally has attempted to access an invalid native web element reference.

System action: The error is handled when a valid reference is obtained automatically. The message is displayed in the console window, but typically does not cause playback failure.

User response: If playback fails and this error message is displayed, contact support.

CRFCN0012E

Test object name is not included in the *script definition* script definition

Explanation: The specified test object name is missing in the script.

System action: Playback fails.

User response: Insert the test object in the script.

CRFCN0013E

The *jvm_name* JVM name is not defined.

Explanation: The specified JVM is not enabled. The JVMs must be enabled for recording and playing back tests on certain applications.

System action: Record or playback while testing applications such as terminal-based applications might not work correctly.

User response: Configure the JVM in the Enable Environments for Testing window. For more information, see the *Enabling Java environments* help topic.

CRFCN0014E

The specified JVM name is an empty string.

Explanation: JVM name cannot be empty strings.

System action: JVM is not enabled for functional testing.

User response: Specify null to reset the current JVM or specify a defined JVM name. Type a valid JVM name in the Enable Environments for Testing window.

CRFCN0015E

The browser *browser_name* is not defined.

Explanation: The browser is not enabled for functional testing.

System action: The applications that are loaded in the specified browser cannot be tested.

User response: Configure the browser for functional testing in the Enable Environments for Testing window. For more information, see the *Enabling web browsers* help topic.

CRFCN0016E

The browser name specified was an empty string.

Explanation: Invalid browser name. Specified browser names must cannot be empty strings and must be valid.

System action: The Browser is not enabled for functional testing.

User response: Specify null to reset the current browser or specify a defined, valid browser name.

CRFCN0017E

The *java_environment* Java environment is not enabled.

Explanation: The specified Java environment is not enabled for functional testing.

System action: The default Jave SE Runtime Environment (JRE) is used during playback. Playback behavior might not be as expected.

User response: Configure the required Java environment in the Enable Environment for Testing window. For more information, see the *Enabling Java environments* help topic

CRFCN0018E

The *line_number* line of the *script_name* script contains an exception.

Explanation: The specified line in the code contains exception.

System action: Playback fails.

User response: Resolve the exception and try the operation again.

CRFCN0019E

The *script_name* script contains the exception on line *line_number*.

Explanation: The specified line in the script contains exceptions.

System action: Playback fails.

User response: Resolve the exception and try the operation again.

CRFCN0020W

No dataset has been initialized for this script

Explanation: datasets must be created and enabled for use in scripts.

System action: dataset actions are not performed.

User response: Create a dataset and initialize it. For more information see the *Creating a dataset* help topic.

CRFCN0021E

The *key_string* keystring is not a valid character or tool name for an unprintable character.

Explanation: The input keys are not valid or might not be supported by Rational® Functional TesterHCL OneTest™ UI.

System action: Record and playback fails.

User response: Type a valid input key.

CRFCN0022E

The *mouse_move* mouse event is not a valid low-level mouse action.

Explanation: An invalid mouse action event or event has not been recorded. Valid mouse action events are right-click, left-click, middle-click, scroll or mouse move.

System action: Record or playback fails.

User response: Try the mouse action event again.

CRFCN0023E

A script asset cannot be renamed to the reserved word, *reserved_word*.

Explanation: Reserved words cannot be used for the naming the script asset.

System action: The script asset name is not changed.

User response: Use a name that is not reserved.

CRFCN0024E

The test object cannot be renamed to this name that is already in use: *object_name*

Explanation: A test object name must be unique and cannot be reserved words.

System action: The test object name is not changed.

User response: Use a unique name for the test object.

CRFCN0025E

The specified test object name does not exist: *object_name*

Explanation: The test object is missing in either the test object map or the script.

System action: The test object is not renamed.

User response: Insert the test object in the script or the object map before renaming it.

CRFCN0026W

A file already exists with the name of the proposed new project log folder: *folder_name*

Explanation: A log folder with the specified name already exists in the location. The new project log in this location must have a different name.

System action: The log folder is not created.

User response: Specify a different name for the log folder or save the log folder in a different location.

CRFCN0010E

Invalid iteration count on playback of script.

Explanation: This message is displayed when invalid iteration count like --1 is typed.

System action: The error is handled by executing the script once.

User response: Type the correct iteration count number.

CRFCN0011E

Error loading the Object Map.

Explanation: The object map is not loaded from the script explorer. This might be because the object map is deleted or does not exist.

System action: Playback stops.

User response: Verify that the object control is present in the script explorer. If the object is not present, add the test object.

CRFCN0029E

Error copying template.

Explanation: The template file cannot be copied from the installation directory to project directory when creating a Rational® Functional TesterHCL OneTest™ UI project. It might be due to not having access or the network might be down.

System action: Operation stops. The error is logged in the log and trace files.

User response: Verify that the template directory exits and the file exits in the Rational® Functional TesterHCL OneTest™ UI installation directory.

CRFCN0036E

Cross project script call to project *project_name* not supported on agent machine.

Explanation: The callscript is not able to download script from the cross project when you use cross project with agent machine.

System action: The error message is logged in the Rational® Functional TesterHCL OneTest™ UI console, the log file and the trace file.

User response: Verify that the project that is called exists and script is accessible. If error persists contact support.

CRFCN0037E

Unable to load script *script_name* from project *project_name*.

Explanation: The script from the datastore is not loaded. The script name might not be complete or the script does not exists due to non existence of the data store.

System action: Playback stops. The error is logged in Rational® Functional TesterHCL OneTest™ UI console, the log, and the trace file.

User response: Verify that the script exists in the project directory. If the error persists, contact support.

CRFCN0038E

Exception occurred during playback of script.

Explanation: This message is displayed during playback. The object not found or class does not happen. For more detail refer the complete error message. This happens when there is some problem running (playback) the functional test script.

System action: A message is displayed an exception occurred during script playback.

User response: Check if the datastore path is correct.

CRFCN0039E

Specified class cannot be found in the project

Explanation: During playback of a test script the specified class cannot be found in the project location specified.

System action: While playing back a script, the corresponding class gets loaded from the project location. If the class file is not present then you get this error.

User response: Ensure that the Rational® Functional TesterHCL OneTest™ UI project is correct and not corrupted. Try building the project again and ensure that the class is recreated on the file system.

CRFCN0042E

Shared dataset could not be loaded

Explanation: The shared dataset is not available. The shared dataset might have been moved or deleted.

System action: Recording stops. An exception is displayed while recording.

User response: Ensure that the shared dataset is available for the script.

CRFCN0043E

Failed to insert script commands into script at suggested location, line count failure.

Explanation: The message is displayed when inserting line outside the class area during recording in an already recorded script. The line count does not match till a line and there are extra information at the end of the cache. There is a difference in the line counting scheme that might be due to non-LineFeed line terminators.

System action: Error message is displayed and the execution continues.

User response: Verify that you insert the script commands within Rational® Functional TesterHCL OneTest™ UI lines in the class.

CRFCN0045E

Please inspect your script file for special characters that may affect line computations.

Explanation: This problem occurs when you insert special characters in the script while recording.

System action: An error dialog box is displayed. However, the recording operation continues.

User response: You must remove special characters in the script.

CRFCN0047E

dataset could not be loaded

Explanation: The dataname is incorrect or the dataset does not exists.Rational® Functional TesterHCL OneTest™ UI cannot load a dataset that is associated with the script while recording.

System action: Recording stops and an error message dialog box is displayed.

User response: Ensure that the dataset exists.

CRFCN0048E

Unable to record while there is an open modal dialog.

Explanation: When a modal dialog box is open, the recorder cannot be activated.

System action: Recording does not start.

User response: Close all open modal dialog boxes, and restart the recorder.

CRFCN0056E

The verification point name contains invalid characters.

Explanation: The verification point name might contain invalid characters such as ', " and so on, and the verification point name cannot be empty.

System action: The verification point is not saved.

User response: Specify a name with valid characters.

CRFCN0064E

The specified dataset iteration count *iteration_count* is not valid.

Explanation: This message is displayed when dataset iteration is invalid.

System action: System will handle by executing the dataset once.

User response: Correct the datapool iteration count during playing back.

CRFCN0073E

Application of unknown type.

Explanation: The selected object is in a domain that Rational® Functional TesterHCL OneTest™ UI does not support. If Rational® Functional TesterHCL OneTest™ UI cannot get a proper name to display the objects in the object map while selecting an object from the object browser, adding the object to the object map or taking verification point, then this error message is displayed.

System action: The system displays an exception. The name displayed in the object map tree is not correct.

User response: No user response required

CRFCN0074E

Could not start the application

Explanation: The application cannot be started as the application configured, is not available at the configured system path specified in the configuration tool.

System action: The system displays an error dialog box with the error id and the error message. Rational® Functional TesterHCL OneTest™ UI does not start the application.

User response: Verify that the configured application is located in the path that is specified in the configuration tool.

CRFCN0081E

No Customization directory is specified.

Explanation: The Customization directory is not specified. For Windows the registry settings and for Unix the environment variable is not set correctly.

System action: An exception is logged in Rational® Functional TesterHCL OneTest™ UI debug file.

User response: Depending on your operating system, complete one of these tasks:

- For Windows environment check the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software \Rational Test\8\Rational FT Customization Directory` `HKEY_LOCAL_MACHINE\Software \HCL Technologies\HCL OneTest UI\HCL FT Customization Directory` customization directory and verify that it is set properly.
- For the Unix environment check the environment variable RATIONAL_FT_CUSTOMIZATION_DIRECTORY HCL_FT_CUSTOMIZATION_DIRECTORY is properly set. Contact support if the problem persists.

You can also check the error details in the debug file. To enable the debug file:

1. Open the ivory.properties file located at `<Functional Tester installation directory>\SDP \FunctionalTester\bin` `<HCL OneTest UI installation directory>\HCLOneTest \FunctionalTester\bin`
2. Set `rational.test.ft.debug.enabled=true`
3. Set `rational.test.ft.debug.filter=default,1`
4. Save the file and open Rational® Functional TesterHCL OneTest™ UI.

CRFCN0082E

Error in RecognitionAttributes format error.

Explanation: Rational® Functional TesterHCL OneTest™ UI has attempted to find a domain implementation section and another child element.

System action: The system displays an exception.

User response: Verify that the customization and .rftop files contain only domain implementation child elements that are present at the same level.

CRFCN0094E

The subitem *subitem_name* is not valid for a Find Property Set.

Explanation: When the correct subitem such as atDescendant(), atChild(), atPrpoerty() and so on is not provided in the dynamic find() method.

System action: The playback stops and the exception is displayed in Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Pass the valid subitem parameters in the find() method to resolve the exception.

CRFCN0096E

Unable to locate entry in map with a null map id.

Explanation: During playback when Rational® Functional TesterHCL OneTest™ UI is unable to find control and read from object map as the object ID is null .

System action: Playback stops.

User response: Add that control that could not be found to the test object.

CRFCN0104E

Errors encountered processing source files in Object Map merge operation.

Explanation: Rational® Functional TesterHCL OneTest™ UI is either unable to merge object maps or is not able to save the merged map.

System action: Object maps cannot be merged.

User response: Verify that the object maps to be merged exists. If the error persists, contact support.

CRFCN0105E

Clipped screen rectangle is empty.

Explanation: When a clipped rectangle of the control is empty, such a situation indicates that the control is completely clipped by a clipping parent. Because the clipped rectangle is empty, Rational® Functional TesterHCL OneTest™ UI cannot perform input actions such as click and so on. The clipped rectangle returns the clipped-screen rectangle for the associated control relative to the top-left corner of the screen. Although getScreenRectangle() returns the true rectangle of the control, this method clips the true screen rectangle to the bounds of any clipping parent control. Clipping parents such as scroll panes or a browser window, may hide part or all of the true screen rectangle. Only the screen-level viewable rectangle is returned, null if the object is not showing, or is completely clipped by a clipping parent.

System action: Playback stops and the message is displayed in Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Ensure that the control is not completely clipped by its clipping parent. If you are taking a screen snapshot, make sure that the control is not clipped by its parent at all.

CRFCN107E

Unexpected test data type in dynamic VP

Explanation: The dynamic verification point cannot be processed. This might occur if the verification point file has been manually edited.

System action: Playback stops and an exception is displayed in the log.

User response: Verify that the tag formats in the verification point file are correct.

CRFCN0110E

InvokeTimeoutException raised for method on object id

System action: The system displays an exception. Play back stops and an exception is created in the log file and IDE.

User response: In the **Maximum time to attempt to find Test Object** field, increase the default timeout for Rational® Functional TesterHCL OneTest™ UI. Click **Window > Preference > Functional Test > Playback**.

CRFCN0113E

Cannot construct a registered object

Explanation: A registered object cannot be constructed when Rational® Functional TesterHCL OneTest™ UI associates a registered object with the Application Under Test. The registered objects fails to get constructed if there is an installation problem or Rational® Functional TesterHCL OneTest™ UI internal error.

System action: The system displays an exception.

User response: No response is required.

CRFCN0114E

Construct valuemanager failed while trying to add it to the test system.

Explanation: The system encountered an installation error or Rational® Functional TesterHCL OneTest™ UI internal error.

System action: The system displays an exception.

User response: No user response required.

CRFCN0115E

Construct property converter failed while trying to add it to the test system.

Explanation: The system encountered an installation error or Rational® Functional TesterHCL OneTest™ UI internal error.

System action: The system displays an exception.

User response: No user response required.

CRFCN0116E

Construct value converter failed while trying to add it to the test system.

Explanation: The Construct value converter fails when Rational® Functional TesterHCL OneTest™ UI instantiates the Value converter on the Application Under Test. The system encountered an installation error or Rational® Functional TesterHCL OneTest™ UI internal error.

System action: The system displays an exception.

User response: No user action required

CRFCN0117E

Construct display value class failed while trying to add it to the test system.

Explanation: The Construct display value fails when Rational® Functional TesterHCL OneTest™ UI creates the display in the Application Under Test for Software Widget Toolkit and Java Swing based applications. The system encountered an installation error or Rational® Functional TesterHCL OneTest™ UI internal error.

System action: The system displays an exception.

User response: No action required

CRFCN0118E

Unable to open file

Explanation: This problem occurs while reading the XML file. The error message displays the name of the file. If this file is not present in the physical location, then this error message is displayed.

System action: The system displays an exception. This exception is displayed along with the other exceptions while reading Rational® Functional TesterHCL OneTest™ UI script assets. For example: ObjectMap, DatastoreDefinition.

User response: Verify that the XML file exists in the file system, is correct and well formed.

CRFCN0120E

Could not locate the data file

Explanation: This problem occurs while reading the XML file. The error message displays the name of the file. If this file is not present in the physical location, then this error message is displayed.

System action: This exception is displayed along with the other exceptions while reading Rational® Functional TesterHCL OneTest™ UI script assets. For example: ObjectMap, DatastoreDefinition.

User response: Verify that the XML file exists in the file system, is correct and well formed.

CRFCN0121E

Unprocessed objects from XML file

Explanation: The XML file cannot be read as some elements are not parsed. The XML file is either corrupted or not well formed.

System action: This exception is displayed along with the other exceptions while reading Rational® Functional TesterHCL OneTest™ UI script assets. For example: ObjectMap, DatastoreDefinition.

User response: Verify that the XML file is correct and well formed.

CRFCN0122E

Extra base elements in XML file

Explanation: The XML file cannot be read as some elements are not parsed. The XML file is either corrupted or not well formed.

System action: This exception is displayed along with the other exceptions while reading Rational® Functional TesterHCL OneTest™ UI script assets. For example: ObjectMap, DatastoreDefinition.

User response: Verify that the XML file is correct and well formed.

CRFCN0140E

Unable to resize window to specified dimensions *dimension1*, *dimension2*.

Explanation: When the values specified for window width and height are not valid, the window cannot be resized.

System action: Playback stops and the message is displayed in Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Specify valid values for the resize operation. If the problem persists, contact support.

CRFCN0141E

Invalid argument to window move operation.

Explanation: The window cannot be moved because the *point_argument* value that is provided to the Point argument is invalid.

System action: Playback stops and the message is displayed in Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Specify valid values for the move operation. If the problem persists contact support.

CRFCN0142E

Unable to move window to specified location: *location1*, *location2*.

Explanation: The *x-coordinate* and *y-coordinte* values being passed as an argument to Point might not be valid. Values cannot be null.

System action: Playback stops and the exception will be shown in Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Specify valid values to be passed to the Point argument.

CRFCN0144E

Cannot capture mouse: mouse already captured.

Explanation: When a call attempts to capture the mouse that is already captured by the Mouse method. Only one mouse action can be captured at a time is allowed. If the mouse action is already captured at the time of a Mouse.capture() call, the message is displayed.

System action: Playback stops and the message is displayed in the Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Capture the mouse action once at a time.

CRFCN0145E

LD_PRELOAD not set correctly

Explanation:

System action:

User response:

CRFCN0151E

A visible property set method could not be located for Property: *property_name*: Class: *class_name*.

Explanation: The setproperty method has been called on an object that does not have this property or cannot set this property.

System action: Playback stops and the message is shown in the log.

User response: Playback stops and the message is shown in the log.

CRFCN0152E

Cannot get screen rectangle for subitem.

Explanation: No action can be performed on a subitem that is not visible. When a subitem is hidden the screen rectangle cannot be determined.

System action: The playback stops and the error message is displayed in the Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Contact IBM Software Support.

CRFCN0153E

Cannot get screen point for subitem.

Explanation: A test is playing back against a subitem and the screen point for clicking the subitem cannot be located.

System action: The playback stops and the message is displayed in the Rational® Functional TesterHCL OneTest™ UI console and in the playback log.

User response: Contact Support.

CRFCN0160E

Error getting option information from the customization file.

Explanation: Option information cannot be obtained from the customization file when there is a problem with the Rational® Functional TesterHCL OneTest™ UI customization file which is related to options or RecognitionProperties section.

System action: The controls in the application under test is not recognized.

User response:

- Verify that the *.rftcust file is present in the `Document and Settings\All Users\Application Data\IBM\RFT` `C:\Users\ADMIN\AppData\Roaming\HCL\HOTUI` and IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR. Verify that the options section is present and correct.
- Verify that the *.rftop file is present in `Document and Settings\All Users\Application Data\IBM\RFT` `C:\Users\ADMIN\AppData\Roaming\HCL\HOTUI` and IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR. Verify that RecognitionProperties section is present and correct.

CRFCN0161E

Error getting proxy information from the customization file.

Explanation: Proxy information cannot be obtained when there is a problem with the Rational® Functional TesterHCL OneTest™ UI customization file related to the proxies section.

System action: The controls in the application under test is not recognized.

User response: Verify that the *.rftcust file is present in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization` and in the Rational® Functional TesterHCL OneTest™ UI installation directory. Verify that the proxies section is present and correct.

CRFCN0162E

Error getting property converter information from the customization file

System action: The controls in the application under test is not recognized.

User response: Verify that the *.rftcust file is present in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization`. Verify that propertyConverters section is present and correct.

CRFCN0163E

Error getting value manager information from the customization file

Explanation: This problem occurs when Rational® Functional TesterHCL OneTest™ UI IDE or the playback or recorder processes are started.

System action: The system displays an exception.

User response: Verify that the *.rftcust file is present in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization`. Verify that propertyConverters section is present and correct.

CRFCN0164E

Error getting value converter information from the customization file

Explanation: This problem occurs when Rational® Functional TesterHCL OneTest™ UI IDE or the playback or recorder processes are started.

System action: The system displays an exception.

User response: Verify that the *.rftcust file is present in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization` and `RFT_INSTALL_DIR` `HOTUI_INSTALL_DIR`. Verify that valueConverters section is present and correct.

CRFCN0165E

Error getting role information from the customization file

Explanation: This problem occurs when Rational® Functional TesterHCL OneTest™ UI IDE or the playback or recorder processes are started.

System action: The system displays an exception.

User response: Verify that the *.rftcust file is present in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization`, and verify that the rolesMap section is present. Verify that the roleMap node is correct in XML data.

CRFCN0166E

Error getting configuration information from the configuration file

Explanation: Configuration information cannot be obtained when there is a problem with the *.rftcfg file. These configurations are related to the environment enabled or the configured applications for testing. This problem occurs during playback or while opening the Application Configurator wizard.

System action: The system displays an exception.

User response: Verify that the Rational® Functional TesterHCL OneTest™ UI configuration file is correct and contains the section configurations.

CRFCN0168E

Error getting image library information from the customization file

Explanation: Third-party software that is used for an image verification point or for optical character recognition can cause problems. This problem occurs when Rational® Functional TesterHCL OneTest™ UI IDE or the playback or recorder processes are started.

System action: The system displays an exception.

User response: Check the Rational® Functional TesterHCL OneTest™ UI configuration file and make sure the path of the third-party software is correct. Verify that the *imageVPLibrary* or *ocrVPLibrary* node, if present in the configuration file, is set correctly.

CRFCN0169E

Error loading configuration file

Explanation: This problem occurs when you try to use the Configure Application tool or Enable Environment tool.

System action: Rational® Functional TesterHCL OneTest™ UI cannot open the Configure Application tool or Enable Environment tool. You may not be able to add application using the application configuration tool. While recording the application is not displayed, in the application list.

User response: Verify that the Rational® Functional TesterHCL OneTest™ UI configuration file is correct. Try restarting Rational® Functional TesterHCL OneTest™ UI by closing all the applications that Rational® Functional TesterHCL OneTest™ UI recorded or played back.

CRFCN0170E

Error storing configuration file

Explanation: Saving after making changes in the Configure Application tool, Enable Environment tool, or Configure Object Recognition tool without write privileges, can cause errors.

System action: Rational® Functional TesterHCL OneTest™ UI does not save the changes made using the configuration wizard in the configuration files.

User response: Verify that you have permissions to write the customization and configuration file present in `C:\ProgramData\IBM\RFT` `C:\ProgramData\HCL\HOTUI`.

CRFCN0182E

Value Out Of Range: *value* not in *minValue* to *maxValue*.

Explanation: The *value_number* value is out of range. The entered value must be within *minValue* to *maxValue*.

User response: Specify a value within the given range, and try the operation again.

CRFCN0185W

Invalid numeric range specified: *value*

Explanation: A null range was specified by this *value*. For example, the function to get updated object, the two arguments that have to be passed are object and display. If the display is an instance of PropertySheet, but its Propertyset is null or display is an instance of TextEditor or TextComparator but the object is not an instance of NumericRange this exception can occur.

User response: Specify a valid range and try the operation again.

CRFCN0192E

The verification point baseline file is not available.

Explanation: The baseline file for the verification point is missing.

System action: Playback stops and an exception is displayed in the log.

User response: Verify if the baseline file for the verification point `<script name>.<vp name>.base.rftvp` exists in the resources folder.

CRFCN0193E

Cannot record dynamic VP baseline when not in interactive mode: need to set option -rt.interactive true

Explanation: The **rt.interactive** option is not set to true in the command line.

System action: Playback does not start

User response: Use the `rt.interactive=true` option in the command line.

CRFCN0194E

Verification Point file does not exist

Explanation: Baseline file for the verification point is missing. This might occur during playback or when you try to open the verification point editor

System action: Operation fails

User response: Verify if the baseline file `<script name>.<vp name>.rftvp` for the verification point exists in the resources folder. Also, verify if the full path for the baseline file is valid

CRFCN0195E

Dynamic Verification Point cannot update script definition

Explanation: The script definition file is open or no permissions to update the script definition file.

System action: Record operation fails

User response: Verify whether the corresponding script definition file is not open and is writable.

CRFCN0200E

Error displaying help file *file_name*.

Explanation: When you click Help or press F1 to view the help content, the Help is not displayed.

System action: The requested help topic is not displayed. An error message is displayed with the name of the Help file that did not open along with the exception stack trace and message.

User response: Verify that you are connected to the internet if online help option is selected. If not, verify that the help was downloaded and installed. If the error persists, contact IBM Software Support.

CRFCN0203E

Highlight failed: Test object name not found in script: *object_id*.

Explanation: When highlighting an object in the application under test, the highlighted object is not found in the script.

System action: Playback stops.

User response:

To resolve this issue, try one of these actions:

1. Start the application under test from the Application Configuration Tool
2. Open the script with which the test object is associated

CRFCN0209E

Exception: *className*: *exception*

Explanation: The type is not supported. To be displayed, class names and file types must be supported.

User response: Specify a supported file type, and then try the operation again.

CRFCN0210W

No data selected for comparison. Select the checkboxes of the nodes you wish to test. Click YES to continue, NO to cancel.

Explanation: No data is selected for comparison. A comparison requires that comparison data be selected.

User response: Select the check boxes of the nodes to test. Click **Yes** to continue with no data for comparison. Click **No**, select data for comparison, and then continue.

CRFCN0211W

Do you want to keep a copy of the existing baseline?

Explanation: This message is asked before the verification point is updated with the changes made.

User response: Decide if a copy of the existing baseline must be kept.

CRFCN0213W

Unable to access assets for script: *scriptName* : *className*

Explanation: The script definition file could not be found. The script cannot be played back if the assets cannot be found. The permission level denied the user access to the script assets. The correct user permission level is required to access these assets.

System action: The operation to get script description fails.

User response: Verify that the script definition (scriptname.rftdef) file exists in the workspace. If the file exists, check the permission level and try the operation again. If the problem persists contact IBM Software Support.

CRFCN0215W

Test Object Map has been changed. Save changes?

Explanation: This is a prompt for user to save the changes to the test object map.

User response: Decide whether to save changes or not.

CRFCN0216W

Do you really want to overwrite an existing Object Map?

Explanation: This is a prompt to confirm while overwriting an existing object map.

User response: Decide whether to overwrite changes or not.

CRFCN0219W

The map editor could not be hidden before displaying the Insert Test Object wizard opened.

Explanation: The top-level window has not changed. This message notifies you of the situation.

System action: The top level window was not changed to display the insert object wizard.

User response: The map editor is still the top level window. Close the map editor before starting the insert object wizard.

CRFCN0225W

Unable to display help about new Test Objects in the Object Map

Explanation: Help is not available for this feature.

System action: The help on adding new test objects in the object map is not displayed.

CRFCN0226W

Requested Test Object not in Map: Map id: *mapId*

Explanation: The test object might not belong to the script. The requested test object is not included in the *mapId* map.

System action: The operation to set selected test object fails.

User response: Select a valid test object and try the operation aga. If the error persists, contact support.

CRFCN0227E

The source node can not be a child of the target node.

Explanation: When you drag an object within an object map, the object is moved within the hierarchy that it is currently part of. Moving an object to a higher level within its hierarchy is not permitted.

System action: The current operation stops.

User response: Do not move the object to a higher level within the hierarchy that it is already a part of.

CRFCN0228E

The target node can not be a child of the source node.

Explanation: When you drag an object within an object map, the object is moved within the hierarchy that it is currently part of. Movement of an object to a lower level within its hierarchy is not permitted.

System action: The current operation is aborted.

User response: Do not move the object to a lower level within the hierarchy that it is already part of.

CRFCN0232W

Verification Point has been changed. Save changes?

Explanation: The verification point has been changed and must be saved to keep the changes.

User response: Decide whether you want to save the changes to the verification point.

CRFCN0234E

The text editor cannot be edited while it is showing hidden characters.

Explanation: Data verification point is created for a control and **Show Hidden Characters** option is selected for the text in the text editor of verification point wizard. No editing is allowed in this mode when you modify text in the text editor while this option is selected.

User response: Disable **Show Hidden Characters** mode and try the operation again.

CRFCN0238W

No test object selected to update recognition properties from.

System action: No test object has been selected for updating recognition properties.

User response: Select a test object and try the operation again.

CRFCN0239W

Be aware that updating an object with a different type of object is typically an error.

System action: An attempt was made to update an object of one type with another object of another type. Only objects of the same type can be used for updating.

User response: Use objects of the same type for updates.

CRFCN0240E

No test objects that meets the search criteria were found.

Explanation: When you click **Find and Modify** option in the object map, the search criteria does not match any of the objects in the object map.

System action: No search result is displayed for the search criteria.

User response: Modify search criteria and try the search again.

CRFCN0245E

The type of value in line {lineNumber} is not correct

Explanation: The type of value in line line_number is not correct. When you modify the value of a property in the object map, a valid value must be used. A valid value is a non-null value.

System action: Validate action row failed.

User response: Verify the type of value in the line, and try the operation again.

CRFCN0246E

.class property cannot be removed or changed weight

Explanation: The .class property of an object cannot be removed and the weight of the .class property of an object cannot be changed. Only value of the .class property can be changed. You can change the value or weight of other properties such as .classindex.map.ui.find.quick.stringempty

System action: Validate action row failed. The original .class property and its weight are retained.

User response: Do not remove the .class property or its weight.

CRFCN0247E

The search could not be completed.

Explanation: The quick find function requires string that are not empty.

System action: No results are displayed.

User response: Type a string that contains characters as a search for the quick find function.

CRFCN0248W

You are now in Pause mode. Click the **Resume** toolbar button to resume browsing the application.

Explanation: The operation performed in pause mode will not be recorded.

System action: The operation to perform action fails.

User response: Click **Resume** in the toolbar to resume the recording.

CRFCN0249E

The *file_name* help file cannot be displayed.

Explanation: The help information cannot be displayed because it is read-protected.

System action: The help file could not be displayed.

User response: Verify the status of the file to make sure that it is not read-protected. If the file is not read-protected, try opening the help information again. If the problem persists, contact support.

CRFCN0251W

To capture data for Windows and Net apps you must first hover over a window in the desired application and select the SHIFT key.

Explanation: This is a warning on capturing data in Windows and .Net applications.

System action: The data is not captured properly.

User response: Hover over the window in the desired application and select the SHIFT key.

CRFCN0252E

Error updating method names in script.

Explanation: An internal error prevented renewing the name of the object in the scripts that the object is associated with.

System action: The script is not updated.

User response: Close the wizard and reopen. Try the operation again. If the error persists, contact support.

CRFCN0253E

The page_name wizard page cannot be created.

Explanation: The page name is either null or has zero length. To be valid, wizard page names must contain characters.

System action: The wizard page is not created.

User response: Type a valid page name.

CRFCN0254E

Configuration directory variable not set in registry. Changes will not be saved. Do you want to continue?

Explanation: The configuration directory variable values in the registry is modified.

System action: Changes in the application configuration tool is not saved if **Yes** option is selected. But you can continue to use the tool and run the application from this tool. The application configuration tool is closed if **No** option is selected.

User response: Try to reinstall the product. If the problem persists contact support.

CRFCN0255E

The RATIONAL_FT_CONFIGURATION_DIRECTORY environment variable is not set or has been changed. If you continue, your changes will not be saved. Do you want to continue?

Explanation: The variable values of the configuration directory in the registry are not set or have been changed.

System action:

- If you click **Yes**, the changes in the application configuration tool is not saved. You can continue to use the tool and run the application from this tool using the already existing setting.
- If you click **No**, the application configuration tool is closed.

User response: Reinstall Rational® Functional TesterHCL OneTest™ UI. If the problem persists contact support.

CRFCN0256W

You are about to set a disabled JRE as the default. Are you sure you want to continue?

Explanation: The JRE that you want to set as default has been disabled.

System action: Displays the message and waits for user response.

User response: Decide whether you want to continue with the activity or not.

CRFCN0257W

You are about to set a disabled browser as the default. Are you sure you want to continue?

Explanation: The browser that you want to set as default has been disabled.

User response: Decide whether you want to continue with the activity or not.

CRFCN0258W

Are you sure you want to remove the default JRE?

User response: Decide whether you want to continue with the activity or not.

CRFCN0259W

Are you sure you want to remove the default browser?

User response: Decide whether you want to continue with the activity or not.

CRFCN0260W

You are about to enable a currently unsupported browser. This is not recommended, and may result in browser corruption. Are you sure you want to continue?

User response: Decide whether you want to continue with the activity or not.

CRFCN0261W

Are you sure you want to disable the default JRE?

User response: Decide whether you want to continue with the activity or not.

CRFCN0262W

Are you sure you want to disable the default browser?

User response: Decide whether you want to continue with the activity or not.

CRFCN0264E

JVM configured already. The JVM [jvmName] is at the same location.

Explanation: A JVM at the specified location is already configured.

System action: The JVM is not added

User response: Specify another JVM or to configure a new JVM, specify the location and try the operation again.

CRFCN0265E

Browser configured already. The browser [browserName] is at the same location.

Explanation: A browser at the specified location is already configured.

System action: The browser is not added.

User response: Specify another browser or to configure a new browser, specify its location and try the operation again.

CRFCN0266E

The name cannot be changed as specified.

Explanation: Names cannot be empty-strings.

System action: The old name is retained.

User response: Specify a name with characters.

CRFCN0267E

A JVM named [jvmName] already exists - retaining the old JVM

Explanation: A JVM by the same name already exists for the project.

System action: The old JVM name is retained.

User response: Specify another JVM.

CRFCN0268E

The name field cannot be empty. The current value, *name*, is retained.

Explanation: The **Name** field cannot be empty for a browser, Java Virtual Machine, or an application while enabling the environments.

System action: The name is changed. The current name is retained.

User response: Specify a valid browser, Java Virtual Machine, or an application name.

CRFCN0269E

A browser is already named. The current browser and name are retained. *browser_name*.

Explanation: Browser name is used to identify the browser, so the name must be unique.

System action: The current name and browser are retained.

User response: Specify a unique browser name.

CRFCN0270E

An Eclipse platform is already named. The current platform and name are retained. *eclipse_platform*.

Explanation: Instances of the Eclipse platform must have unique names. The current platform is retained because its name is specified. Eclipse Platform name must be unique.

System action: The current name is retained.

User response: Specify a unique name for the platform.

CRFCN0271E

Errors encountered while starting browser.

Explanation: In Rational® Functional TesterHCL OneTest™ UI you can verify whether the web browser is enabled. The application attempts to start the browser, but it cannot be started for the enablement test.

System action: The enablement test for the browser fails.

User response: Verify that the browser is installed correctly and that you can start it from a command line. If you cannot start the browser from a command prompt, try to reinstall it.

CRFCN0273W

There is already another instance of the Enabler running. More than one Enabler instance cannot be started.

Explanation: Rational® Functional TesterHCL OneTest™ UI allows only one instance of the enabler to run.

System action: Rational® Functional TesterHCL OneTest™ UI does not allow you to launch another enabler.

User response: You must ensure that only one instance of the enabler is running.

CRFCN0274W

The Application Configuration tool and Enabler cannot be run simultaneously.

Explanation: One of the operations must be completed before starting another. Therefore the Object Properties Configuration tool and Enabler cannot run at the same time.

System action: The launch of one of the application is cancelled, as no two applications can run at the same time

User response: You must ensure that only one application is running at a time.

CRFCN0275W

The Object Properties Configuration Tool and Enabler cannot be run simultaneously.

Explanation: One operation should be completed before starting the other.

System action: Either of them is launched based on your response.

User response: You can decide which operation must be opened or closed.

CRFCN0276E

JVM_name is not a file or directory in a Java environment. The Java Virtual Machine (JVM) cannot be enabled.

Explanation: The specified file or directory does not contain a JVM, so the JVM cannot be enabled.

System action: No JVM is enabled

User response: Specify the name of a valid file or directory that contains a JVM.

CRFCN0279E

The *JVM jvmName* cannot be enabled until after the system is rebooted.

Explanation: Java Virtual Machine (JVM) cannot be enabled until system reboot. Prohibiting the JVM from being enabled until after the computer is restarted prevents some required files from being deleted.

System action: The JVM was not enabled .

User response: Restart the computer, and then try to enable the JVM.

CRFCN0280E

Error enabling JVM *JVM_name*: *exception_message*.

Explanation: This is the error message that you get when you enable a Java Environment (JVM). The error message states that the JVM cannot be enabled.

System action: The Java virtual machine (JVM) cannot be enabled. Java applications using this JVM cannot be tested.

User response:

- Verify that the user has Administrator privileges.
- Verify that the JVM bin directory has write permission for the user.
- Verify that the JVM is installed correctly.

CRFCN0281E

Failed to get install directory.

Explanation: When you enable your environment for testing, the Rational® Functional TesterHCL OneTest™ UI installation directory is not found.

System action: The environment is not enabled for testing.

User response:

To resolve the directory location problem, perform these actions:

- Verify that Rational® Functional TesterHCL OneTest™ UI is installed correctly. If the software application is not installed correctly, reinstall it.
- Verify that the environment variable *IBM_RATIONAL_RFT_INSTALL_DIR HCL_INSTALL_DIR* is set correctly.

CRFCN0282E

A valid JVM could not be found at path *jvmPathname*.

Explanation: The specified path does not contain the required JRE

System action: JRE is not enabled.

User response: Verify the JRE path, and try to enable the JRE again.

CRFCN0283E

Internal error (invalid JVM object)

Explanation: An internal error occurs if the configuration file is corrupted. The JVM is not enabled if the configuration file is corrupted.

System action: The enable operation fails.

User response: You must configure JVM again to resolve this problem.

CRFCN0284W

Disabling the product's JRE is not recommended. Are you sure you want to disable this JRE?

Explanation: Disabling the product's JRE is not recommended.

System action: The computer waits for you to decide whether to disable the JRE.

User response: You must decide if you want to disable the JRE or not.

CRFCN0285E

Error disabling JVM *JVM_name*: *exception_message*.

Explanation: This error message is displayed when you disable the Java environment.

System action: Java virtual machine (JVM) is not fully disabled.

User response:

- Verify that the Java directory has write permissions for the user.
- Restart the computer, and then enable and disable JVM.

CRFCN0286W

A few of the enablement files used by the running JVM will be removed after the system restarts. You must not enable the JVM after a system restart.

Explanation:

System action: JVM is disabled.

User response: Click OK. Restart the system and enable JVM again.

CRFCN0288E

fileName file cannot be found among the installation files.

Explanation: The specified file cannot be found due to installation problems

System action: The installation could not be completed because the JVM file is not enabled correctly.

User response: You must reinstall the application to obtain the missing files in the installation directory.

CRFCN0289E

The accessibility.properties file could not be read.

Explanation: The accessibility.properties file is read protected.

System action: The JVM is not enabled.

User response: Ensure that the file is not read protected.

CRFCN0291E

Cannot find Rational® Functional TesterHCL OneTest™ UI installation directory

Explanation: The environment variable *IBM_RATIONAL_RFT_INSTALL_DIR* HCL_HOTUI_INSTALL_DIR does not exist or the variable path is not a valid Rational® Functional TesterHCL OneTest™ UI installation directory.

System action: Enable environment operation fails

User response: Set the environment variable *IBM_RATIONAL_RFT_INSTALL_DIR* HCL_HOTUI_INSTALL_DIR and point the full path of the Rational® Functional TesterHCL OneTest™ UI installation directory.

CRFCN0292E

Error enabling browser.

Explanation: This message is displayed when not able to enable the browser to perform the recording and playing back action.

System action: You are prevented from recording and playing back any action against the browser in the HTML domain.

User response: Check the troubleshooting section to enable the browser. If still the problem persists, contact support.

CRFCN0294E

Error enabling third-party browser extensions.

Explanation: Internet Explorer third party extensions must be enabled for theRational® Functional TesterHCL OneTest™ UI browser helper to run.

System action: Internet Explorer third-party extensions is not enabled.

User response: Check the troubleshooting section in Rational® Functional TesterHCL OneTest™ UI Help to enable the browser. If still the problem persists, contact IBM software support.

CRFCN0296E

The browser at *browser_path* directory cannot be disabled. Reason: *error_message*

Explanation: For testing, the browser must be disabled. Disabling the browser requires administrator privileges.

System action: The browser is not disabled and testing cannot continue.

User response: Verify that you are logged in with administrator privileges. If you are logged in with administrator privileges and the problem persists, contact support.

CRFCN0299E

Cannot modify installed-chrome.txt.

Explanation: This message is displayed when enabling or disabling the browser.

System action: Rational® Functional TesterHCL OneTest™ UI will not be able to enable or disable the plug-ins properly from the browser. To enable or disable the browser, the installed-chrome.txt file must be modified. Administrator privileges are required to modify this file.

User response: Verify that you are logged in with administrator privileges. If you are logged in with administrator privileges and the problem persists, contact support.

CRFCN0300E

Cannot write overlays.rdf.

Explanation: To enable or disable the browser, the overlays.rdf file must be saved. Administrator privileges are required to save the file.

System action: Rational® Functional TesterHCL OneTest™ UI will not be able to enable or disable the plug-ins from the browser. If the plugin is not properly enabled then Rational® Functional TesterHCL OneTest™ UI will not be able to record or playback the actions.

User response: Verify that you are logged in with administrator privileges. If you are logged in with administrator privileges and the problem persists, contact support.

CRFCN0302E

A browser is required to have a name

Explanation: A browser must have a name.

System action: Unable to create a browser object with the specified name, as it is not a valid name . The invalid name might be null or the length of name might be zero.

User response: Specify a valid name for the browser.

CRFCN0303E

The *file_name* file is not the specified directory for an installed browser.

Explanation: Either no browser is installed or an unsupported browser is installed in the specified path.

System action: Unable to add a browser from the path specified.

User response: Select a path that contains a supported and installed browser.

CRFCN0304E

Search directory does not exist.

System action: No search results are displayed.

User response: Specify a valid directory to search.

CRFCN0305E

The *enable_File* cannot be modified.

Explanation: Writer privileges are required to modify the file. The user must have these privileges.

System action: The modify operation cannot be completed.

User response: Verify that you have write privileges for the file to be modified. Access to the enabler directory is required to change the file.

CRFCN0306E

An Eclipse shell is required to have a name

Explanation: An exception is thrown if no Eclipse name is specified.

System action: Invalid Eclipse exception is displayed.

User response: Specify a valid Eclipse shell name.

CRFCN0307E

The *folder_name* directory is not the directory of an installed Eclipse shell.

Explanation: The specified directory must contain an Eclipse shell installed.

System action: The directory was not approved. Unable to add a JVM from the path specified.

User response: Specify a valid directory that contains an installed Eclipse shell.

CRFCN0308E

The *file_name* file could not be opened. *Exception_message*.

Explanation: Appropriate permissions are required to open the file. The security settings of the log file prevents from opening the file.

System action: The HTML log file does not open.

User response: Check the permissions and try the operation again.

CRFCN0309E

Error creating verification point display.

Explanation: After a verification point fails during script playback, an HTML log is displayed. However, if the Java SE Runtime Environment (JRE) is earlier than version 1.3.2_02, the HTML log cannot be displayed.

System action: A message is displayed that the JRE associated with the browser should be of version 1.3.1_02 or later for the verification point comparator to be displayed.

User response: Install JRE version 1.3.1_02 or later. Enable this JRE, and retry displaying the log. For more information on enabling JRE, see Enabling Java environments topic and for enabling web browsers, see Enabling web browsers topic in Help.

CRFCN0310E

Has been commented out in FtIdMessages.properties

Explanation:

System action:

User response:

CRFCN0317E

The TSS log could not be opened.

System action: The TestScriptServices (TSS) log could not be opened.

User response: Specify the build, log folder and log name using: setLogBuild(String logBuild), setLogFolder(String logFolder) and setLogName(String logName) respectively.

CRFCN0321E

The log file could not be written to. The *event_code* code is unknown.

Explanation: The event code specified does not correspond to the standard set of events. Event codes must correspond to the standard set of events.

System action: The write to log file operation fails.

User response: Contact support.

CRFCN0325E

The *TSS_measure* timer could not be started. The *exception_code* exception is reported.

System action: The timer does not start. Playback fails.

User response: See the troubleshooting information regarding the exception and try the operation again. If the operation fails, contact support.

CRFCN0326E

The *TSS_measure* timer could not be stopped. The *exception_code* exception is reported.

System action: The timer does not stop.

User response: See the troubleshooting information regarding the exception and try the operation again. If the operation fails, contact support.

CRFCN0327E

The *button_icon* toolbar image could not be created. *exception_message*.

System action: The toolbar button is not created.

User response: Follow the troubleshooting in the error message.If the operation fails, contact support.

CRFCN0328E

The *file_name* file could not be saved. *exception_message*.

Explanation: Appropriate permissions are required to save the file.

System action: The text is not saved to the file.

User response: Check the file permissions and try the operation again.

CRFCN0329E

The message cannot be displayed correctly. *message_level*.

Explanation: The error level or kind of message specified is not in standard format. For example, if the msgKind_variable is set to 0, only the error messages is displayed. If the msgKind_variable is set to 2, even the information messages is displayed. This error might occur due to internal errors.

System action: The message is not displayed.

User response: Specify a valid error message level or type in the Monitor Message Preferences in the recorder monitor.

CRFCN0330E

Error trying to output to the monitor *exception_message*.

Explanation: An internal generic method to log error messages is called while trying to output script-related messages during recording or playback.

System action: The error message is written to the log file.

User response: No action is required.

CRFCN0331E

The Test Manager data store cannot be accessed.

Explanation: Logging into the Rational project is required for access to the Test Manager datastore for the project.

System action: Could not retrieve the datastore path. The getProjectDatastorePath() could be called from a number of process including script playback.

User response: Log into the Rational project with appropriate credentials.

CRFCN0337E

The *source_name* script source was not created in the current Rational project.

Explanation: In a project, script source names must be unique. The specified name of the script source is used.

System action: The project is not connected.

User response: Specify a different logical name and try the operation again.

CRFCN0342E

The *file_name* file could not be opened. *exception_message*.

Explanation: Appropriate permissions are required to open this file. The security settings of the file prevents from opening the file.

System action: The file is not opened.

User response: Verify that the user has the appropriate permissions and try the operation again. If the problem persists, contact IBM Software Support.

CRFCN0343E

Exception {result}

Explanation: This message is displayed when the type of event is general and there is a property name to describe the name of an exception.

User response: The response should be based on the type of exception. For example, events like Script Start, Configuration and so on. If the problem persists, contact support.

CRFCN0345W

The log already exists. Overwrite?

System action: Waits for the user response.

User response: Decide whether to overwrite the existing log or create a new log.

CRFCN0346W

Unable to remove the asset files for script *scriptname*. You can remove the script asset files using the Navigator view.

System action: Delete operation fails.

User response: Remove the script asset files using the Navigator view.

CRFCN0347W

Deleting *mapname* object map will make following *projectname* script(s) unusable. Do you want to delete *mapname*?

System action: Wait for the response.

User response: Decide whether you want to delete the map name or not.

CRFCN0348W

Deleting *dataset* dataset will make following *projectname* script(s) unusable. Do you want to delete *dataset*?

Explanation: The dataset that is associated with the script(s) is mentioned and deleting the dataset makes the script unstable.

User response: Decide whether you want to delete the dataset or not.

CRFCN0349W

Project *project1* is associated with Rational project *project2*. If you delete this project, the corresponding script source in the Rational project will become invalid. Would you like to proceed?

System action: Wait for user response.

User response: Decide whether you want to delete the associated project or not.

CRFCN0351E

Could not create project

System action: You cannot create a Rational® Functional TesterHCL OneTest™ UI project. Record and Playback options does not work.

User response: Check that you have privileges to write to the folder where you are creating the project. If you have privileges, try to create the project again.

CRFCN0352E

The project could not be connected to.

Explanation:

The project is not Java based. This message is displayed in these instances:

- The subtype of the datastore project does not match the runtime type. Datastore project subtypes and runtime types must match for connections to be established.
- FtInstallOptions.ALLOW_VBNET_REMOTE_TESTING is not true.

This indicates that the current setting has to be a Java project only. The valid project subtype is JAVA_PROJECT_SUBTYPE.

User response:

Do one these steps, and then try the operation again:

- Use a Java project.
- Verify that the datastore subtype and the runtime type match.
- Make FtInstallOptions.ALLOW_VBNET_REMOTE_TESTING true in ivory.properties file.

CRFCN0353E

Could not connect to the project

Explanation: Projects that are created with a newer version of the product cannot connect to projects created with an older version of the product.

System action: You cannot connect to the project.

User response: Ensure that the project is created with a newer version of Rational® Functional TesterHCL OneTest™ UI. Verify that the project directory contains the project and the datastore definition is present.

CRFCN0354W

This project was created with an older version product which has been upgraded with major change. Click **Yes** to upgrade the project and connect, click **No** to abort. Warning: After upgrade, previous versions of the product will not be able to connect to the project. If the project is under Source Control, this operation will modify the server copy as well.

System action: Displays the message and waits for response.

User response: Decide whether to upgrade or not.

CRFCN0355W

This project was created with an newer version product. Incompatibilites could arise. Would you like to continue anyway?

User response: Decide whether to continue with the operation or not.

CRFCN0356W

This project was created with an older version product which has been upgraded with minor change. Click **Yes** to upgrade the project and connect, click *No* to continue.

System action: Displays the message and waits for user response.

User response: Decide whether to upgrade the project or not.

CRFCN0357W

Test suite support may not work properly without upgrading the project!

Explanation: This message is displayed after the project is created with an older version product that has been upgraded with minor change.

System action: Displays the message and waits for user response.

User response: Decide whether to upgrade the project or not.

CRFCN0358E

No expected VP found

Explanation: The expected verification point file in the log folder is missing.

System action: Verification point comparator is not opened

User response: Ensure that the expected verification point file exists in the log folder.

CRFCN0359E

The Object Map associated with script does not exist.

Explanation: This message is displayed when the object map does not exist. Object file is either deleted, moved or corrupted

System action: Playback stops and an exception is displayed.

User response: Ensure that the object map exist.

CRFCN0360W

You cannot start recording while the Application Configuration Tool is open. Do you want to close the Application Configuration Tool and start recording?

System action: Displays the message and waits for user response.

User response: Close the Application configuration tool befroe you start recording.

CRFCN0361W

You cannot start the Enabler while the Application Configuration Tool is open. Do you want to close the Application Configuration Tool and start the Enabler?

System action: Displays the message and waits for user response.

User response: Close the Application configuration tool before starting the Enabler.

CRFCN0362W

You cannot start the Object Properties Configuration Tool while the Application Configuration Tool is open. Do you want to close the Application Configuration Tool and start the Object Properties Configuration Tool?

System action: Displays the message and waits for user response.

User response: Close the Application configuration tool before starting the Object properties configuration tool.

CRFCN0363E

The script cannot be loaded.

Explanation: The script is not loaded and so the script cannot be edited, copied, or run.

System action: System will show an error and will continue with null exception.

User response: Check that the script has a script definition file. The script definition file has a .rftdef format. The location of the file is *<workspace folder>\<project_name>\resources*.

CRFCN0364W

Couldn't find Test Object in application-under-test. Application may not be running or enabled, or your environment may not be enabled. Test Object may not be visible. Would you like to configure application for testing?

System action: Displays the message and waits for user response.

User response: Check that the application and environment are enabled, and the application is running. If not, configure and run the application.

CRFCN0365W

You cannot start recording while the Enabler is open. Do you want to close the Enabler and start recording?

System action: Displays the message and waits for user response.

User response: Close the Enabler before you start recording.

CRFCN0366W

You cannot start the Application Configuration Tool while the Enabler is open. Do you want to close the Enabler and start the Application Configuration Tool?

Explanation:

System action: Displays the message and waits for user response.

User response: Close the Enabler before starting the Application Configuration Tool.

CRFCN0367W

You cannot start the Object Properties Configuration Tool while the Enabler is open. Do you want to close the Enabler and start the Object Properties Configuration Tool?

System action: Displays the message and waits for user response.

User response: Close the Enabler before starting the Object properties configuration tool.

CRFCN0368W

You cannot start recording while the Object Properties Configuration Tool is open. Do you want to close the Object Properties Configuration Tool and start recording?

System action: Displays the message and waits for user response.

User response: Close the Object properties configuration tool before you start recording.

CRFCN0369W

You cannot start the Application Configuration Tool while the Object Properties Configuration Tool is open. Do you want to close the Object Properties Configuration Tool and start the Application Configuration Tool?

System action: Displays the message and waits for user response.

User response: Close the Object properties configuration tool before you start the Application configuration tool.

CRFCN0370W

You cannot start the Enabler while the Object Properties Configuration Tool is open. Do you want to close the Object Properties Configuration Tool and start the Enabler?

System action: Displays the message and waits for user response.

User response: Close the Object properties configuration tool before you start the Enabler.

CRFCN0371E

The command-line value *command_value* for the *option_variable* option is not a valid number.

Explanation: The command-line value that is specified for the option is not in the standard format. The standard format consists of type string, boolean, integer, double, long or float.

System action: The correct value is not set.

User response: Type the command-line value for the option in correct format, and retry the operation.

CRFCN0372E

The *option_name* option cannot be found.

Explanation: The specified option does not exist. Depending on the scenario specify a valid option name. For example :DEBUG_ENABLED is a valid option name.

System action: The correct value is not set.

User response: Type the option name in a supported format. The valid options are in .rftcust file in *FunctionalTest\bin* folder.

CRFCN0373E

Error reading the configuration file

System action: Record or playback stops with an exception. Playback stops and an error box is displayed. Recorder stops with a message written to the log and trace file.

User response: Verify that the Rational® Functional TesterHCL OneTest™ UI configuration file is present in `C:\ProgramData\IBM\RFT\configuration` `C:\ProgramData\HCL\HOTUI\configuration` and in the Rational® Functional TesterHCL OneTest™ UI installation directory.

CRFCN0374E

The compile operation could not be completed.

Explanation: When constructing a ScriptCompile object, neither the script nor the language were specified. If the script is specified, the language is the same as specified by the script definition language. If an command that is similar to `compile all` is run, the language must be specified. Depending on the scenario, either the script name or the language must be specified.

System action: The compile operation stops.

User response: Specify the script name or the language depending on the compile scenario.

CRFCN0375E

File not found: *file_name*. The script *script_name* cannot be compiled.

Explanation: The *definition_filename* script definition file for the *script_name* script file that defines the language cannot be found. In scenarios where the script file is specified, the language is specified in the script definition file. If the script definition file is not found, the script cannot be compiled.

System action: The compile command does not run.

User response: Verify that the script definition file with .rftdef format exists in the *<workspace folder>\<project_name>\resources* folder and try the operation again.

CRFCN0376E

The *command_name* Java compiling command cannot be run. The Java error message follows: *JavaError_message*.

Explanation: A compile-time error occurred while compiling the script file in Java.

System action: The compile command fails.

User response: Try correcting the Java error. If the error persists, contact support.

CRFCN0377E

Executing Java compiler command [{0}]\nExit code:\n[{1}]

Explanation: The command returns a non-zero exit code.

System action: Compilation fails because of some Java syntax errors.

User response: You must check for Java syntax errors.

CRFCN0379E

Load script class failed [script class] [msg]

Explanation: The class fails while loading the script.

System action: The system displays a ClassNotFound Exception

User response: You must verify if the class exists and then load the script again.

CRFCN0380E

Construct script class failed.

Explanation: The class file of script is either not created, missing or deleted. This happens when script resources like rftdef file is missing.

System action: Playback stops and an exception is displayed.

User response: Ensure that all script resources exist and the required libraries are added to the path.

CRFCN0381E

If either -{0} or -{1} options are specified, then both must be specified.

Explanation: Information for groups like user name and project information is obtained from the command line. If any of the information is not complete, then an invalid command line exception is displayed.

System action: The input validation fails.

User response: You must enter absolute information for groups like user name and project information.

CRFCN0382E

Error occurred while trying to create the record toolbar

Explanation: Problem encountered when the recording begins.

System action: Script cannot get recorded if this exception is displayed.

User response: Try starting the recording again. If the problem persists, contact support.

CRFCN0383E

-{0} must be followed by a path

Explanation: In keyword section, when the option specified is either DATASTORE, RMT_PROJECT or GENIE_PROJECT the path also must be specified.

System action: An InvalidCommandLineException is displayed.

User response: A valid path must be provided.

CRFCN0384E

Only one -{0} allowed in the command line arguments

Explanation: Only one command line argument is allowed. All option pairs (name/value) start with a hyphen (-). Any option not starting with a hyphen is an argument to the command preceding it on the command line (playback) when the option is DATASTORE and datastore != null.

System action: An InvalidCommandLineException is displayed.

User response: Support must be contacted.

CRFCN0385E

-openscript command line option must be followed by a script name

Explanation: The openscript command line option must be followed by a script name.

System action: The openscript operation fails.

User response: You must specify a valid script name after the openscript command

CRFCN0386E

-line must be followed by a line number

Explanation: The line command must be followed by a line number.

System action: The line operation fails.

User response: You must specify a valid, non-negative line number after the line option

CRFCN0387E

Line number must be a positive integer

Explanation: The line command must be followed by a positive integer.

System action: The line operation fails if a negative value is entered.

User response: You must specify a valid, positive integer for line number.

CRFCN0388E

-openhelp command line option must be followed by a help topic

Explanation:

System action:

User response:

CRFCN0389E

The IDE specified "{0}" is not supported

Explanation: WSWPlugin and Visual Studio IDE's are supported by Rational® Functional TesterHCL OneTest™ UI.

System action: The operation that requires IDE's different from the ones supported by Rational® Functional TesterHCL OneTest™ UI fails.

User response: You must verify that the operations carried out are supported by the WSWPlugin and Visual Studio IDE.

CRFCN0390E

Executing rational_ft command failed

Explanation: This error message is displayed when Rational Test Manager does not launch the Functional Test script. The message also has the command that is used to launch the script.

System action: The test script is not launched and the error message is displayed in the Rational Test Manager.

User response:

- If the command in the error message is a Java script, ensure that the JVM arguments are correct. The JVM arguments can be modified through the option rational.test.ft.client.jvm_options in *product installation directory*`\ivory.properties` file.
- If the command in the error message is a Vb.net`\rational_ft.exe` `\hcl_ft.exe` script, ensure that *product installation directory* is present. If the file is not present, the .Net framework might have not been installed or installed after Rational® Functional TesterHCL OneTest™ UI is installed.
- Ensure that the project does not end with backward slash.

CRFCN0391E

Executing rational_ft command [{0}] Error message [{1}]

Explanation: An error occurs while executing the rational_ft command.

System action: The operation fails.

User response: Try resolving the problem based on the error message that is displayed or contact support.

CRFCN0392E

-startide must precede -openhelp <help-topic>

Explanation: An Integrated Development Environment (IDE) must be started before accessing Help topics.

System action: The help topic does not open.

User response: You must start the IDE and then access the help topics.

CRFCN0393E

-startide must precede -openscript <script-name>

Explanation: A script opens only in an Integrated Development Environment.

System action: The open script command fails.

User response: You must start the IDE before opening the script.

CRFCN0394E

-startide must precede -recordscript

Explanation: You can record a script only in a running Integrated Development Environment (IDE).

System action: The record script command fails.

User response: The IDE must be started before creating a script.

CRFCN0395E

-startide must precede -createscript

Explanation: A script can be created only in a running Integrated Development Environment (IDE).

System action: The create script command fails.

User response: The IDE must be started before creating the script.

CRFCN0396E

-startide must precede -activate

Explanation: An Integrated Development Environment (IDE) must be started in order to activate it.

System action: Activate command fails.

User response: You must start the IDE, before performing the activate operation.

CRFCN0397E

No datastore specified

Explanation: A datastore must be specified to initiate a session. Session can be any process with separate process id, For example: Playback.

System action: The session initiation fails.

User response: You must specify a datastore, and then initiate a session.

CRFCN0398E

Invalid line number [{0}] specified. Must be greater than or equal to zero (0).

Explanation: Line number cannot be a negative value.

System action: The recording of a script fails.

User response: You must specify a positive integer as the line number.

CRFCN0399E

No file name specified

Explanation:

System action:

User response:

CRFCN0400E

No file name specified

Explanation: A file name must be specified while editing or displaying a file.

System action: Edit file operation failed.

User response: You must specify an editable file name.

CRFCN0401E

No map name specified

Explanation: To create an object map a map name must be specified.

System action: The create map operation fails.

User response: You must specify a valid map name.

CRFCN0402E

No actual VP file name specified

Explanation: A file name for the verification points must be specified.

System action: The compare verification operation fails.

User response: You must specify a valid verification point file name.

CRFCN0403E

Combine Maps: No target map file name specified

Explanation: Combine the specified object maps and optionally update the specified script definitions to use the target object map.

System action: The combine maps operation fails.

User response: You must specify a valid target map file name.

CRFCN0404E

Combine Maps: No source script definition or map file names specified

Explanation: The source script definition or map file names specified

System action: The combine maps operation fails.

User response: You must specify a valid target map file name.

CRFCN0405E

Enabler does not have a JRE or browser by the name [{0}]

Explanation: The enabler must have a valid JRE or a browser name specified.

System action: The enable operation fails.

User response: You must provide a valid JRE or browser name and then try enabling the JRE or the browser again.

CRFCN0406E

A failure occurred trying to write the config for the executable.

Explanation: This error message is displayed when Rational® Functional TesterHCL OneTest™ UI is not able to create the config file for the executable so that testing assembly can be used with it.

System action: The system displays an error message.

User response: You must check if assembly can be run using .NET 1.0 framework.

CRFCN0407E

The Java system property specifies Rational® Functional TesterHCL OneTest™ UI install directory that cannot be found.

Explanation: The installation directory that the IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR environment variable specifies does not exist.

System action: Record and playback fails.

User response: Specify the correct installation directory in the IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR environment setting.

CRFCN0408E

The installation directory specified in the system environment variable is null.

Explanation: The IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR variable must include the correct path information for the installation directory.

System action: Record and playback fails.

User response: Specify the correct installation directory in the IBM_RATIONAL_RFT_INSTALL_DIR HCL_HOTUI_INSTALL_DIR environment setting.

CRFCN0409E

-startide command line option must be followed by an idetype: wswplugin or vsdotnet

Explanation: You must specify the type of IDE to be started.

System action: The Start IDE operation fails.

User response: You must specify an IDE type after startide command.

CRFCN0410E

-openscript command line option must be followed by a script name

Explanation: A script name must be specified to open the script specified in the openscript command.

System action: The Open script command fails.

User response: You must specify a valid script name after the command.

CRFCN0411E

-{0} must be followed by a help topic

Explanation: The openhelp option must be followed by the help topic you want.

System action: The open help command fails.

User response: You must specify a help topic.

CRFCN0412E

Only one -{0} allowed in the command line arguments

Explanation: The command line allows only one argument. All option pairs (name/value) start with a hyphen (-). Any option not starting with a hyphen is an argument to the command preceding it on the command line (playback) when the option is DATASTORE and datastore != null

System action: The command fails.

User response: You must specify only one option at a time.

CRFCN0413E

The -{0} directory [{1}] can not be found.

Explanation: The directory in which the datastore is located cannot be found.

System action: The command fails.

User response: You must provide a valid directory.

CRFCN0414E

-{0} command line option must be followed by a script name, or follow an option which specifies the script name

Explanation: Command line option must be followed by a script name.

System action: The command fails.

User response: You must specify a script name or options that imply a script name.

CRFCN0415E

-{KeywordRecordOption}-{0} command line option must be followed by a keyword, RMT Datastore, keyword file and keyword script name

Explanation: A command line option must be followed by a keyword, RMT Datastore, keyword file and keyword script name.

System action: The system does not execute the command.

User response: You must specify the keyword, RMT Datastore, keyword file and keyword script name after the KeywordRecordOption option

CRFCN0418E

There is no data after the -{0} command line option

Explanation: There is no data after the command line option.

System action: The script fails to execute.

User response: You must specify the argument values.

CRFCN0419E

-{0} option only valid after a -playback option. It must follow a playback option and before the next record option.

Explanation: In a command line exception, valid operation is to follow the instruction in the error message, For example: Execute the option only after -playback.

System action: The command fails to execute.

User response: You must perform only valid operations.

CRFCN0420E

There is no project path after the -{projectPathOption} command line option

Explanation: Project path is not specified.

System action: The command fails to execute.

User response: You must specify the required project path.

CRFCN0421E

-{enableNameOption} option must be followed by a name of a browser or JRE configured in the Enabler.

Explanation: -{enableNameOption} option must be followed by a name of a browser or must be JRE configured in the Enabler.

System action: The command fails to execute.

User response: You must specify the name of the JRE configured or a browser.

CRFCN0422E

-{displayOption} option must be followed by a filename

Explanation: The file to be displayed must be specified.

System action: The command fails to execute.

User response: You must specify the filename to be displayed.

CRFCN0423EE

-{editOption} option must be followed by a filename.

Explanation: The file to be displayed must be specified.

System action: The command fails to execute.

User response: You must specify the filename to be displayed.

CRFCN0424E

Expected/actual/baseline option must be followed by expected and actual RFTVP filenames

Explanation: The actual and the expected verification point file names are missing in the command-line.

System action: Comparator is not opened. An invalid command line option with the error ID and message is displayed.

User response: Specify the expected and actual verification point file names in the command.

CRFCN0426EE

-{createOption} option must be followed by a script name.

Explanation: The file to be displayed must be specified.

System action: The command fails to execute.

User response: You must specify the filename to be displayed.

CRFCN0427E

-{createMap} option must be followed by a map filename.

Explanation: A map file name must be specified to create a map file.

System action: The command fails to execute.

User response: You must specify the map filename.

CRFCN0428E

-{combineMaps} option must be preceded by a -map mapFileName option.

Explanation: The maps to be combined must be specified.

System action: The command fails to execute.

User response: Specify the -map mapFileName option.

CRFCN0429E

-{helperOption} option must be followed by a script name.

Explanation: The file to be displayed must be specified.

System action: The command fails to execute.

User response: You must specify the filename to be displayed.

CRFCN0430E

-{createHelperSuper} option must be followed by a full class name specification.

Explanation: -{createHelperSuper} option must be followed by a full class name specification.

System action: helperSuper class name is not created.

User response: You must specify a full class name and try the operation again.

CRFCN0431E

-{CLIENT_MAILSLOT}option must be followed by a mail slot name

Explanation: An error message is displayed while parsing the command line arguments when the option is CLIENT_MAILSLOT but a mailslot name is not specified

System action: The command execution failed.

User response: You must specify a mail slot name and try the operation again.

CRFCN0432E

-{IDE_TYPE} option must be followed by the IDE type

System action: The command execution fails.

User response: You must specify either WSWPlugin or Visual Studio IDE type.

CRFCN0433E

-session_host must be followed by a hostname

Explanation:

System action: The command fails to execute.

User response: Specify the required hostname.

CRFCN0434E

-session_port must be followed by a port number

Explanation:

System action: The command fails to execute.

User response: You must specify the required port number.

CRFCN0435E

-session_id must be followed by a session id number

Explanation: No session ID number was specified. The *- session_id* variable must be followed by a session ID number.

User response: Specify the required session ID number and try running the command again.

CRFCN0437E

The *-passwordOption* option must be followed by a password.

Explanation: A password did not follow the *-passwordOption* option. This option must be followed by a password.

User response: Specify the password after the option and run the command again.

CRFCN0438E

The *-TM_PROJECT* option must be followed by a full path and project file.

Explanation: A full path and project file did not follow the *- TM_PROJECT* option. This option must be followed by a full path and project file.

User response: Specify the full path and project file after the option and run the command again.

CRFCN0439E

The *-LOG_BUILD* option must be followed a build name.

Explanation: A build name did not follow the *- LOG_BUILD* option. This option must be followed a build name.

User response: Specify a build name after the option and run the command again.

CRFCN0440E

The *-LOG_FOLDER* option must be followed the name of a log folder.

Explanation: A log folder name did not follow the *- LOG_FOLDER* option. This option must be followed by the name of a log folder.

User response: Specify the log folder name after the option and run the command again.

CRFCN0441E

The *-LOG_NAME* option must be followed by a log name.

Explanation: A log name did not follow the - *LOG_NAME* option. This option must be followed by a log name.

User response: Specify a log name after the option and run the command again.

CRFCN0442E

-*RESET_OPTION* option must be followed by an option name

Explanation: An option name did not follow the - *RESET_OPTION* option. This option must be followed by an option name

User response: Specify the required option name and run the command again.

CRFCN0443E

The -*UPDATESCRIPTDEFS* option must be followed by a boolean value (true or false).

Explanation: A Boolean value did not follow the - *UPDATESCRIPTDEFS* option. This option must be followed by a Boolean value (*true* or *false*).

User response: Specify the Boolean value as `True`to update the script definitions or specify `False` to maintain the script definitions unchanged. Run the command again with the required Boolean value.

CRFCN0444E

The -*MAP/FROMMAP* option must be followed by a datastore relative map file name.

Explanation: A datastore relative map file name did not follow the - *MAP/FROMMAP* option. This option must be followed by a datastore relative map file name.

User response: Specify the datastore relative map file name after the option and run the command again.

CRFCN0445E

The - option must be followed by a datastore relative dataset file name.

Explanation: A datastore relative dataset file name did not follow the - *option_name* option. This option must be followed by a datastore relative dataset file name.

User response: Specify a datastore that is relative to the dataset file name and run the command again.

CRFCN0446E

The -*ITERATION_COUNT* option must be followed by an iteration count

Explanation: An iteration count did not follow the - *ITERATION_COUNT* option. This option must be followed by an iteration count.

User response: Specify an iteration count after the option and run the command again.

CRFCN0447E

The *-ITERATION_COUNT* option must be followed by a valid iteration count: *value* is not valid use ALL or a positive value

Explanation: The *- ITERATION_COUNT* option must be followed by a valid iteration count; the *value* count is not valid. A valid iteration count is `ALL` or a positive integer. Iteration count cannot be negative or greater than the available iterations.

User response: Specify a valid iteration count and run the command again.

CRFCN0448E

The *-script* option must be followed by a full script name.

Explanation: A full script name did not follow the *- script* option. This option must be followed by a full script name.

User response: Specify a full script name after the option and run the command again.

CRFCN0449E

The *INSERTAFTER* option has an invalid line number: *value*

Explanation: The *value* line number associated with the *INSERTAFTER* option is not valid. The line number specified cannot be negative or greater than the number of lines.

User response: Specify a valid line number after the option and run the command again.

CRFCN0450E

-CREATE_TESTOBJECT option must be followed by a script name.

Explanation: A script name did not follow the *- CREATE_TESTOBJECT* option. This must be followed by a script name.

User response: Specify a script name after the option and run the command again.

CRFCN0451E

-option command line option must be followed by a script name, or follow an option which specifies the script name

Explanation: A script name was not specified for the *- option* command-line option. This command-line option must be followed by a script name or the command-line option must follow an option that specifies a script name.

User response: Specify the script name after the option or include an option that specifies the script name before the *- option* command-line option.

CRFCN0452E

-insertionPoint command line option must be preceded by an insertion point, use option *-INSERTBEFORE* or *-INSERTAFTER*

Explanation: An insertion point or use option did not precede the - *insertionPoint* command-line option. This option must be preceded by an insertion point or a use option, - *INSERTBEFORE* or - *INSERTAFTER*

User response: Specify an insertion location after the option and run the command again.

CRFCN**0454**E

Option *option_argument* has no value

Explanation: The *option_argument* argument has no value. This argument must have a value.

User response: Specify a valid value, and run the command again. See the rational.rftcust file in `Functional Tester/bin` folder for valid options.

CRFCN**0455**E

Log format *log_format*is not a known log format; must be in *valid_log_format*

Explanation: The log *log_format* format is not a known format. The valid format is expressed this way: *valid_log_format*

User response: Follow the valid log format, and run the command again.

CRFCN**0456**E

Command line argument *argument* considered to be an argument for the last *-playback* option because it does not match anything else. But no *-playback* option has been specified on the command line prior to this.

Explanation: No *-playback* option has been specified on the command line prior to this.

System action: Playback fails and an invalid command line exception is thrown.

User response: Specify an option for the command line such as *-playback*, and run the command again.

CRFCN**0457**E

An application is required to have a name

System action: Displays InvalidApplication exception.

User response: Specify a valid application name.

CRFCN**0458**E

appName is not a valid application

Explanation: The application is not supported by Rational® Functional TesterHCL OneTest™ UI.

User response: Specify a valid application and try the operation again.

CRFCN0461E

excMsg Default information will be loaded.

System action: Internal error on running configure application tool. The changes are not saved and the default information is loaded.

User response: Specify a valid operation and try loading the information again. If the issue persists, contact support.

CRFCN0462E

Error getting application information

Explanation: While opening the application configuration tool, the configuration details could not be found.

System action: The configuration tool is not displayed.

User response:

- Verify that the Rational® Functional TesterHCL OneTest™ UI configuration file is present in `C:\ProgramData\IBM\RFT\configuration` `C:\ProgramData\HCL\HOTUI\configuration`.
- Verify that the configuration directory is present in `RFT_INSTALL_DIR` `HOTUI_INSTALL_DIR`. If the installation directory path is incorrect, correct it.

CRFCN0463E

Could not start the application

Explanation: The application name might or specified path might not be correct.

User response: Verify the application name and path, and then try to start the application again.

CRFCN0465E

Empty name field not allowed - retaining old name of *name*

Explanation: A name was not provided in the **Name** field. An application cannot be renamed with a null or empty string.

System action: The previous application name, *name*, is retained.

User response: Provide non-empty string in the **Name** field, and try to rename the application again.

CRFCN0466E

An application named *value* already exists - retaining original name *oldname*.

Explanation: An *app_name* application name already exists. The application name must be unique.

System action: The previous application name, *oldname*, is retained.

User response: Specify a unique application name and try to name the application again.

CRFCN0468W

There is already another instance of the Application Configuration Tool running.

Explanation: Another instance of the Application Configuration Tool is already running. Only one instance of the application configuration tool can run at a time.

System action: The new instance is not started.

User response: Close the first instance and start the tool again or wait until the first instance has run.

CRFCN0469W

The Enabler and Application Configuration Tool cannot be run simultaneously.

System action: The Enabler and Application Configuration tool cannot be run simultaneously. The Enabler or the Application Configuration tool is running.

User response: Make sure that one action is completed, and then start the other action.

CRFCN0470W

The Object Properties Configuration Tool and Application Configuration Tool cannot be run simultaneously.

Explanation: One of the tools is already running. The Object Properties Configuration tool and Application Configuration tool cannot run simultaneously.

System action: Only one tool will run at a time.

User response: Make sure that one action is completed, and then start the other action.

CRFCN0471E

AppConfigTool.mainFromDialog() caught exception: *exception*

System action: The previous application that was selected is shown.

User response: Try to rectify based on the exception shown. If the error persists, contact IBM Software Support.

CRFCN0472E

Cannot find or read the application information.

Explanation: The configuration tool from the Rational® Functional TesterHCL OneTest™ UI is unable to read the configuration file.

System action: The configuration tool does not open. The Error Dialog box with the error ID and message is displayed.

User response: Verify that the Rational® Functional TesterHCL OneTest™ UI configuration file is present in `C:\ProgramData\IBM\RFT\configuration` `C:\ProgramData\HCL\HOTUI\configuration`.

CRFCN0473E

File read error

Explanation: Permission to read and write the customization file is required

System action: An Error Dialog box with the error ID and the message is displayed

User response:

- Verify that the customization file located in `C:\ProgramData\IBM\RFT\customization` `C:\ProgramData\HCL\HOTUI\customization` has read and write privileges.
- Verify that the configuration directory is present in the `C:\ProgramData\IBM\RFT\configuration` `C:\ProgramData\HCL\HOTUI\configuration`.

CRFCN0476E

Invalid dataset specification to iterator

Explanation: The dataset cannot be found or the dataset contains no data.

User response: If the script uses the dataset, verify that the dataset exists, provide the correct path, and try to run the script again.

CRFCN0477E

Invalid dataset equivalence class *class* specified to iterator

Explanation: The dataset equivalence *class_name* class that was specified to iterator either a negative integer or is greater than the equivalence class count. The number specified to the iterator can neither be negative nor greater the equivalence class count.

User response: Check the equivalence class count, and specify a valid equivalence class to the iterator and try to run the operation again.

CRFCN0478E

Variable name *variableName* not found in dataset

System action: The getCell operation fails.

User response: Specify a valid name and index and try the operation again.

CRFCN0476E

Invalid dataset specification to iterator

Explanation: The dataset cannot be found or the dataset contains no data.

User response: If the script uses the dataset, verify that the dataset exists, provide the correct path, and try to run the script again.

CRFCN0480E

Invalid dataset iterator class name: *class*

Explanation: The *class* dataset iterator class name is not valid. The specified class is not a dataset iterator class name. Valid iterator class names are: random and sequential.

User response: Specify a valid iterator class and try to create the dataset iterator again.

CRFCN0481E

Unable to construct dataset iterator class: *class*

System action: The dataset iterator is not created.

User response: Contact support.

CRFCN0482E

Invalid dataset iterator class, not an iterator: *iterator*

System action: The dataset iterator is not created. Valid dataset iterators are: random and sequential

User response: Specify a valid dataset iterator and try the operation again.

CRFCN0483E

dataset file not found

Explanation: This message is displayed when dataset CSV file is missing.

System action: Playback stops and an exception is displayed.

User response: Ensure that the CSV file exists.

CRFCN0484E

dataset could not be read

Explanation: Rational® Functional TesterHCL OneTest™ UI is unable to parse the dataset file. This might be due to incorrect file format.

System action: Playback stops and an exception is displayed.

User response: Check the dataset file format.

CRFCN0485E

Error persisting a dataset in CSV format: file not found: *fileName*

Explanation: The *fileName* CSV file into which the dataset is to be written cannot be found. The CVS must be created before saving dataset data in the file.

User response: Verify that the CSV file exists at the specified path or create the CVS file and try the operation again.

CRFCN0486E

Exception persisting a dataset in CSV format: *file*

System action: The operation to store as CSV fails.

User response: Try to rectify based on the exception message displayed. If the error persists, contact IBM Software Support.

CRFCN0487E

dataset not found: *null*

Explanation: The specified dataset contains no data. Empty, or null, dataset files cannot be loaded.

User response: Specify a non-null dataset file, and try to load the dataset again.

CRFCN0488E

dataset could not be found: *path*

Explanation: The dataset cannot be found.

User response: Verify that the dataset exists in the specified path, and try the operation again.

CRFCN0489E

Exception serializing dataset: *dataset*

Explanation: The dataset could not be serialized for storage.

User response: Try to rectify based on the exception message displayed. If the error persists, contact support.

CRFCN0490E

Invalid equivalence class index *index* specified in method *methodname*

Explanation: The *index* equivalence class index that is specified in the *methodname* method is not valid. The index value is either negative or greater than the maximum index possible. The index value can neither be negative nor greater than the maximum possible. The maximum value of index is one less than the number of items in dataset.

User response: Specify a valid index and try the operation again.

CRFCN0491E

Invalid variable index *index* specified in method *method_name*

Explanation: The *index* variable index that is specified in *method_name* method is not valid. The index value is either negative or greater than the maximum index possible. The index value can neither be negative nor greater than the maximum possible.

User response: Specify a valid index and try the operation again.

CRFCN0492E

Invalid column specification *columnIndex* in *method*

Explanation: The *columnIndex* column specification in the *method_name* method is not valid. The specified column index is either 0 or greater than the number of variables. The column specification can neither be 0 nor greater than the number of variables.

User response: Specify a valid column index and try the operation again.

CRFCN0494E

Invalid equivalence class specification {0} in {1}

Explanation: The *index* equivalence class specification in the *method_name* method is not valid. The index is either negative or greater than the maximum index. The index can neither be negative nor greater than maximum. The maximum value of index is one less than the number of items in dataset.

User response: Specify a valid index, and try the operation again.

CRFCN0495E

Unable to convert *value* to *toType*

Explanation: Certain values cannot be converted to data types. The value might be null.

User response: Specify a valid value and try the operation again.

CRFCN0496E

Unable to reorder records, reorder list is null

Explanation: The reorder list is empty or cannot be found. Null sets cannot be manipulated.

User response: Specify a non-null record list and try to reorder the records again.

CRFCN0497E

Unable to reorder records, record set contains *value1* not *value2*elements

Explanation: The value specified as the size of the record set is incorrect. The record set contains *value1* element not *value2* elements. The record size must be correct.

User response: Check the record size, specify the correct size, and try to reorder the records again.

CRFCN0498E

Unable to reorder records, record *record_order* in reorder list multiple times

Explanation: The *record_order* record is included in the reorder list multiple times. Records can be included once in the reorder list.

User response: Verify that each record is listed only once in a set, and try to reorder the records again.

CRFCN0499E

Invalid record index *index* in *method*

Explanation: The *index* record index in the *method_name* method is not valid. The index is either negative or greater than the maximum index. The index can neither be negative nor greater than the maximum. The maximum value of index is one less than the number of items in dataset.

User response: Specify a valid index and try the operation again.

CRFCN0500E

Invalid cell specification *index* in *method*

Explanation: The *index* cell specification in *method_name* method is not valid. The index is either negative or greater than the maximum index. The index can neither be negative nor greater than the maximum index. The maximum value of index is one less than the number of items in dataset.

User response: Specify a valid index and try the operation again.

CRFCN0501E

Attempt to access a cell in a record that is not a member of a dataset

Explanation: This problem occurs when the dataset record cell that is not in the dataset is accessed. The dataset cell is either deleted or the dataset is changed.

System action: Playback stops and an exception is displayed.

User response: Ensure that the required cell exist in dataset.

CRFCN0504E

Multiple matches found while looking for a *role* with name "*property*".

Explanation: During playback, multiple matches are found while looking for a mappedObject using the properties.

User response: Change the weight of different properties of that mappedObject and try the operation again. If the problem persists contact support.

CRFCN0505E

Multiple matches found while looking for a *name*.

Explanation: During playback, multiple matches are found while looking for a mappedObject using name.

User response: Change the weight of different properties of that mappedObject and try the operation again. If the problem persists contact support.

CRFCN0506E

It appears that more than one instance of the HTML document *name* is open. Close all duplicate browser windows except for the one being tested and click **OK** to retry.

Explanation: Only one HTML document to be tested can be open during the test.

System action: Waits for your response.

User response: Close all duplicate browser windows except for the one being tested, and click **OK** to try the test again.

CRFCN0507E

It appears that more than one instance of the application is running, containing a window with title *appName*. Try closing instances of the application that are not being tested and click **OK** to retry.

Explanation: More than one instance of the application is running and contains a window with the *appName* title .

User response: Close instances of the application that are not being tested, and click **OK** to try to continue testing.

CRFCN0508E

It appears that the application has more than one identical window open. The window caption is *windowName*. Close all identical windows except for the one being tested and click **OK** to retry.

Explanation: The application has more than one identical window open. The window caption is *windowName*.

User response: Close all identical windows except for the one being tested, and click **OK** to continue testing.

CRFCN0509E

Error while parsing the keyword file.

Explanation: The keyword file cannot found or is not in the correct format. The keyword recording fails. The keyword file extension should be .rmt.

User response: Ensure that the keyword referenced exists and is in the correct format, and try to parse the file again.

CRFCN0510E

The location of the dialog box cannot be determined.

Explanation: An error is displayed while setting the display location for a dialog box next to its parent. While recording a script, you click a button on the dialog box, a new dialog box opens, the location of this new dialog box is calculated and is set relative to the parent dialog box. During this operation of calculating the location and positioning the dialog in the user interface an exception is displayed.

System action: The dialog box location cannot be determined.

User response: Adjust the size of the parent dialog box and try the operation again.

CRFCN 0511E

The *file_name* image file cannot be found.

Explanation: The required image file is missing or in an incorrect location. The image file must be in this folder: `C:\Program Files\IBM\IBMIMShared\plugins \com.ibm.rational.test.ft.graphics_8.1.1.v20091030_1158.jar\com \rational\test\ft\graphics C:\Program Files\HCL\HCLIMShared\plugins \com.ibm.rational.test.ft.graphics_8.1.1.v20091030_1158.jar\com\rational\test\ft \graphics`

System action: The operation stops. If the recorder stop button image is missing, the recorder does not start.

User response: Search for the image file elsewhere on the computer. If you find the file, move the file to the correct folder. If you cannot find the file, contact IBM Software Support Support.

CRFCN0512E

Playback cannot continue.

Explanation: When using the callScript(String *scriptFullName*, Object[] *args*, int *iterationCount*) method in the test script, the *scriptFullName* variable does not extend from com.rational.test.ft.script.RationalTestScript. The class that is represented by the *scriptFullName* variable must extend from com.rational.test.ft.script.RationalTestScript.

User response: Revise the class that is represented by *scriptFullName* such that it refers to a com.rational.test.ft.script.RationalTestScript object, and try to play back the test again.

CRFCN0513E

RemoteProxyReferenceValue.fromStream - An attempt was made to create a test object in the application under test.

Explanation: A test object is a connection point between the test script and a proxy object that connects to the real object in the application under test. Rational® Functional TesterHCL OneTest™ UI verifies that the referenced test context is a client. If an attempt is made to create a test object in the application under test, which is never a client, recording cannot continue.

User response: Make sure there is no attempt to create a test object in the application under test while recording.

CRFCN0514E

No channel was found for {0}. The *method_name* method cannot be called.

Explanation: During playback channels provide a means of executing code that requires the GUI thread affinity. Objects of a method call must be assigned a channel. No channel was found for the referenced object.

System action: An exception is displayed if a channel is not assigned to the object while trying to invoke a method.

User response: Verify that a channel is assigned to an object before calling a method that refers to the object.

CRFCN0515E

Playback has stopped.

Explanation: The stop() method was called. When the stop() method is called from the functional test script, playback stops.

User response: No further action is required. You can remove the stop() method call, which is an API call, from the script to prevent playback from stopping.

CRFCN0516E

Cannot create script helper from null object map.

Explanation: A script helper cannot be created from a null object map. Recording a test script involves creation of a script helper. This helper is created at the end of recording by making use of the script definition and the object map. If the object map is null the script helper cannot be created and hence the recording operation is incomplete.

User response: Ensure that the object map is not null and try to create a script helper again.

CRFCN0517E

The test object cannot be activated or its visibility validated.

Explanation: Test script playback cannot be completed. During playback a GUI test object is encountered and the data corresponding to this object needs to be captured. The GUI test object cannot be activated or its visibility cannot be validated. If either the object does not exist or it is not visible playback fails the verification of expected data against test object . For playback, the GUI test object must be activated or its visibility must be validated.

System action: Verification of expected data against the test object.

User response: Avoid unnecessary interactions with the application UI during playback. Make sure that the application is responsive, and then try the play back of the script again.

CRFCN0518E

The command that is issued on the following line cannot be run: command_line.

Explanation: The playback and recording functions are formatted as commands and are passed to Rational® Functional TesterHCL OneTest™ UI. While processing this command Rational® Functional TesterHCL OneTest™ UI encountered an error. The problem is not a command syntax problem.

User response: The error while processing the command depends on the many variations of the application environment. You must take corrective actions based on the exception message.

CRFCN0519E

Property converter format error. The data cannot be retrieved.

Explanation: During playback property converter objects are retrieved from the XML file. This XML file must contain the expected elements with the *Component_Model* name. The XML file provided this element: XML_element. Although playback continues, running that specific line in the script fails.

System action: Data is retrieved from a supplied input object. The data must be retrieved using the name supplied when the object was persisted. If this name does not match with the expected name then an exception is displayed

User response: Ensure that data is retrieved from the correct input object and try the operation again.

CRFCN0520E

Rational® Functional TesterHCL OneTest™ UI cannot start.

Explanation: The number of attributes on the Element_Tag XML element is not valid. The XML element can have only one attribute.

User response: Make sure that the Element_Tag XML element has only one attribute and restart Rational® Functional TesterHCL OneTest™ UI.

CRFCN0521E

Playback cannot continue.

Explanation: The functional test script called the callScript(RationalTestScript *script*, Object[] *args*, int *iterationCount*) method, but the *script* variable is a null reference. The *script* variable cannot be a null reference.

User response: Make sure that the *script* variable is not a null reference, and try to play back the test again.

CRFCN0524E

The script code cannot be added. The session must be active before adding the script code.

Explanation: A method specification has been added to the code sequence that is being generated. The session must be active before script code can be added.

System action: Attempts to add a code sequence for an inactive session

User response: Ensure that the session is active when adding a code sequence to the script.

CRFCN0527E

Playback cannot continue.

Explanation: There might be multiple instances of a test object found, which often happens when multiple instances of the application under test are running. Playback requires that only one instance of a test object be found.

System action:

Playback stops when it encounters more than one instance of a test object. In this case, *counter_value* instances of the same test object were found. During playback, the test was looking for the *object_name* object. The test found these objects:

- First object: Recognition score: *score_1*, description: *description_1*
- Second object: Recognition score: *score_2*, description: *description_2*

User response: Make that only one instance of the an application is running when playback is started.

CRFCN0531E

The script helper cannot be created from null script definition.

Explanation: A script helper is generated based on a script definition file that reflects the set of nodes in an associated map that is used by an instance of the script. If the script definition is null then the script helper cannot be created.

System action: Attempts to create script helper from null script definition. Script definition is generated dynamically during recording.

User response: Ensure that the script definition is present and try creating a script helper file again.

CRFCN0532E

Exception occurred during playback of automated script

System action: During playback, unexpected interaction with the application under test can cause playback to fail.

User response: Avoid interaction with the application under test during playback.

CRFCN0534E

Cannot start application *application_Name*: Cannot find browser *browser_Name*

Explanation: The application cannot be started because the *browser_Name* browser cannot be found. To test the *application_Name* application, the *browser_Name* browser must be found.

System action: The application under test cannot be started for testing.

User response: Make sure that the specified browser is installed in the correct location and that the application is configured correctly; then try to start testing again.

CRFCN0535E

Playback cannot continue.

Explanation: The limit for exceptions was exceeded in event handlers while calling the *method_name* method on the *object_name* object. This limit exists to prevent infinite loops when handling an event and restarting the method. The maximum number of exceptions per method is *exception_number*.

User response: Make sure that the control on which the method is called is visible before starting playback. If playback problems persist, call support.

CRFCN0540E

The recording could not be stopped.

Explanation: An exception occurred when a request to stop the recorder was issued.

User response: Check the exception message for information about the exception, why the exception occurred, and how to resolve the issue. Try restarting the recorder, which can resolve the problem. If the problem persists contact support.

CRFCN0541E

Unable to generate XML: The test-script recording cannot be completed.

Explanation: While recording a test script, customization information is stored in an XML file. The information about an object is serialized into an XML stream and stored in a specified file. The *file_name* file could not be created, and the recording stopped with this exception: exception_data.

System action: Serialization of an object into an XML stream and storing it in the file specified fails.

User response: Problems with the serialization of customization information is available in the exception stack trace. Review the trace, and make corrections based on the exception description. Then try to record the script again.

CRFCN0542E

Playback cannot continue

Explanation: An exception is displayed while closing an browser that does not respond.

System action: The system displays an exception.

User response: Close the browser manually, and try the operation again.

CRFCN0546E

Processing the command-line command cannot continue.

Explanation: The helper superclass full name variable did not follow the *-option_name* option in the command. The *-option_name* option must be followed by a helper superclass full name variable.

User response: Make sure that the helper superclass full name option is entered after the -helpersuper option while running the command from command-line: -helpersuper<*helper superclass full name*>. For more information see, Command line interface on page 1726.

CRFCN0547E

The added object cannot be recognized during a test run.

Explanation: The property and weight recognition attributes for a class in ObjRecProp.rftop are not configured correctly. To configure the properties correctly, the name tag must precede the weight tag in the file for an object class.

User response: Configure the object class, correctly ordering the property tags, and start Rational® Functional TesterHCL OneTest™ UI again.

CRFCN0549E

The ObjectMap map cannot be stored in the *object_map* file.

Explanation: The *object_map* file was modified while the map was loaded into memory. The map cannot be stored if the *object_map* file is modified while the map is loaded into memory. These methods cannot be used to store a map in this scenario: store(ObjectMap objectMap, java.io.File file,boolean silent) or public static void store(ObjectMap objectMap, java.io.File file).

User response: Make sure that the ObjectMap file is not modified while the ObjectMap map is loaded into memory while trying to store the ObjectMap map in the ObjectMap file by using the store() method call of the ObjectMap class.

CRFCN0551E

Playback cannot continue.

Explanation: The *weight_value* value for the WeightedProperty object that the script specified is not valid. Valid property weight values for the com.rational.test.ft.script.WeightedProperty object are 0–100.

User response: Specify a valid property weight and play back the script again.

CRFCN0552E

The recording could not be stopped.

Explanation: An exception occurred when a request to stop the recorder was issued.

User response: Check the exception message for information about the exception, why the exception occurred, and how to resolve the issue. If debugging is enabled, check the debug log for details about the exception. Restarting the recording might resolve this problem. If the problem persists contact support.

CRFCN0553E

Debugging is enabled, an error was encountered, and the debugging log includes an entry for this error.

Explanation: An internal method call attempted to release a registered object, but was unable to.

User response: No action is required. For more information on the error, see the exception message in the log.

CRFCN0555E

Playback cannot continue.

Explanation: The object map file cannot be found. To play back the script, the object map file is required. The object map file is either missing or corrupted.

User response: Verify that the object map file is present and in the correct location. Open the object map file, and verify that it is not empty. Verify that the XML elements in the opened file have matching start and end tags. Correct tags without a pair. Try to play back the script again.

CRFCN0556E

The application does not start due to the absence of configuration information.

Explanation: The configuration information is found in the following directory:`C:\ProgramData\IBM \RFT\configuration\configurations.rftcfgC:\ProgramData\HCL\HOTUI\configuration \configurations.rftcfg`

User response: You must have the `configurations.rftcfg` file with write permission in the local drive.

CRFCN0557E

Playback cannot start.

Explanation: An attempt was made to play back a script in a terminal service environment, for example, on a remote desktop, but the terminal window on the remote desktop was minimized. To play back scripts in a terminal environment, the terminal window on the remote desktop cannot be minimized.

User response: Make sure that the terminal window is not minimized on the remote desktop, and play back the script again.

Tip: The terminal window does not need to be full-screen; for script playback the window cannot be minimized.

CRFCN0558E

Playback cannot continue.

Explanation: A mapped test object was passed as an argument to the equals method of the com.rational.test.ft.object.interfaces.TestObject class. Mapped test objects cannot be passed as the argument to the equals method for this class.

User response: Make sure that a mapped test object is not passed as an argument to the equals method of the com.rational.test.ft.object.interfaces.TestObject class. For example, if you call the GetChildren() method on a mapped test object to return all the children of that object, the returned children are unmapped test objects.

```
applicationmenuBar().click(atPath("File"));
TestObject to[] = applicationmenuBar().getChildren();
for(int i=0;i<to.length;i++){
System.out.println(applicationmenuBar().equals(to[i]));
// Exception is not thrown here to[i] is an unmappedtestobject
System.out.println(applicationmenuBar().equals(applicationmenuBar()));
// Exception occurs here since applicationmenuBar() is a mappedtestObject
}
applicationmenuBar().click(atPath("View"));
```

CRFCN0561E

Recording cannot continue. Event handlers are nested too deeply.

Explanation: Event handlers are nested too deeply. This error occurs when one event handler calls a test object method that triggers another event. A maximum nesting of *nesting_number* is supported. If the nesting level is exceeded, recording stops.

User response: Avoid recording events on the test object so that event handlers are not nested deeply.

CRFCN0562E

Playback cannot continue.

Explanation: The *url* string could not be parsed or the string does not contain a legal protocol. The string returned from the getURL() method is null during playback.

User response: Make sure that the *url* string that is used as an HREF link is correctly formatted or that a legal protocol is used.

CRFCN0563E

An exception occurs while writing data to the XML stream.

Explanation: An exception is displayed while writing data of the object to XML output stream.

User response: The reason for the failure is present in the exception stack trace. You must take the corrective action based on the description of the exception.

CRFCN0564E

Playback cannot continue.

Explanation: The application under test is either not running or the test environment is not enabled correctly. Playback might stop when the application runs in domains that do not require enablement, such as Java or HTML, but the application is either not running or not enabled correctly.

User response: Make sure that the application is running and that the environment is enabled correctly; then play back the script again.

CRFCN0565E

Script playback cannot be completed as the test object is not in the correct state.

Explanation: The expected test object cannot be found. The script playback fails because the test object is not in the correct state. The *object_name1* test object with the *state_name1* state was not found. This *object_name2* is the possible failing candidate with this *state_name2* incorrect state.

System action: Some HTML applications, browsers state is not loaded completely. This results in the test script failing as the expected test object is not found.

User response: Try playing back the script again.

CRFCN0567E

The maps for the specified data objects cannot be combined into the target map file.

Explanation: There is a call to the *public static void combineMaps*(*String datastore*, *String targetMapName, String*[] *source, boolean updateScriptDefs*) method from ObjectMap class. The source array that represents the source data objects contains the name of a file that does not exist or is not a map file or a script definition file. The *sourceName* source data object must be either a map files or script definition files. Specifying any other type of file is not valid and causes an error.

User response: Make sure that the *sourceName* source data object is valid, and try to combine the data objects again.

CRFCN0570E

Playback cannot continue.

Explanation: The *objectmap* object map cannot be stored in the *outputstream* output stream. The *objectmap* variable or the *outputstream* might be null in the static storeToStream(ObjectMap *objectmap*, OutputStream *outputstream*) method that is called from an ObjectMap class. These variables cannot be null. The *outputstream* might not have been closed before calling this method. The output stream must be closed before calling this method.

User response: Provide values for the variables and make sure that the *outputstream* is closed before calling this method. Try playing back the script again.

CRFCN0574E

Playback cannot be started.

Explanation: A attempt was made to play back the script on a locked workstation. Scripts cannot be played back on locked workstations.

User response: Unlock the workstation, and play back the script again.

CRFCN0578E

The *script_name* script cannot be played back.

Explanation: An unhandled exception occurred while playing back the script. Check the log for details about the exception.

User response: See the playback log for the detailed cause of the problem. After correcting the issues found in the log, try to play back the script again.

CRFCN0580E

The recorder could not be started.

Explanation: An unexpected error prevented the recorder from starting.

User response: Check the exception for details about the unexpected error and how to resolve the problem. Restarting Rational® Functional TesterHCL OneTest™ UI and then starting the recorder might resolve the problem. If the problem persists, contact support.

CRFCN0582E

The test object does not contain a mapped reference.

Explanation: Playback has attempted to obtain a property value for the object map, but the object map reference is not present for the test object.

User response: Ensure that the object map exists for the test object and try playing back the script again.

CRFCN0589E

The recording cannot be completed.

Explanation: See the error message exception report for information about the cause of the problem.

User response: Correct problems that the exception report identifies, and try to record the script again. Restarting the application can resolve some problems. If problems persist, contact support.

CRFCN0590E

Playback cannot continue as the test object passed method must be an object that was found, it cannot be a mapped test object.

Explanation: A test object cannot be found in the application under test. The test object that is passed to the ITestObjectMethodState.setFoundTestObject() method must be an object that is found in the application under test. The object cannot be a mapped test object. Playback attempts to find the object again and modify the state of the method invocation. If the object that is found is not a test object in the application under test but is a mapped test object, then playback stops.

User response: Try executing the statement again.

CRFCN0594E

The *application_name* application in the *working_directory* directory could not be started.

Explanation: The application path and the working directory might have been configured incorrectly in the Application Configuration wizard. If the path information for the working directory is incorrect, the application cannot be started.

User response: Make sure that the correct working directory and correct path are used while configuring the application before testing. Correct the configuration and try to run the test again.

CRFCN0596E

Playback has stopped.

Explanation: A manual script shutdown request was received.

User response: No action is required.

CRFCN0597E

Incorrect usage of command-line option.

Explanation: The `-kwrecord` command-line option must be followed by a keyword, a Rational Manual Tester datastore name, a keyword file, and a keyword script name.

User response: Provide the specified elements after the command-line option and run the command again.

CRFCN0602E

The application cannot be started. The location of the application is not given in the configuration information.

System action: No application location was provided in the configuration information. The application cannot start without this configuration information.

User response: Configure the application with all the required details in the **Configure Applications for Testing** wizard.

CRFCN0604E

Playback cannot continue.

Explanation: The com.rational.test.ft.script.Row table row descriptor is not is not defined as a subitem specification and the getSubitem method calls the descriptor. The table row descriptor must be specified as a subitem specification.

User response: Make sure that the com.rational.test.ft.script.Row table descriptor is defined as a subitem specification so that the getSubitem method can call it. Create a Row() object using the Row(SubItem) contructor instead of another overriden constructor. The getSubitem() call works only with a Row(SubItem) created object.

CRFCN0606E

Unable to dynamically enable the .NET application because it uses an older framework that does not support unification.

Explanation: The .NET application cannot be enabled because the application uses an older framework that is not supported. The application must use a current .NET framework.

User response: You must supply a configuration file or adjust the current one to make the application work with a new .NET framework.

CRFCN0607E

The recording could not be paused.

Explanation: An error prevented the recording from pausing. Many circumstances might have generated this error. See the exception message for details about the error and ways to resolve the problem.

User response: Restarting Rational® Functional TesterHCL OneTest™ UI and then starting the recorder might resolve the problem. If the problem persists, contact support.

CRFCN0608E

Unsupported type, value class required: {0}

Explanation: The object type is not supported for serialization to XML stream. It must be a primitive value or must have a value manager You can specify extensible components such as proxies, test object values, and value managers in an external customization file with the .rftcust extension.

System action: Serialization of an object to XML stream fails as the object type is not supported for serializations.

User response: Ensure that the object type is supported for serialization. The object type must either be primitive or a value manager. For more information see **Customization file** topic in the information centre.

CRFCN0611E

The test cannot be played back.

Explanation: To play back the test, the testing application must be able to read the script definition file. The script definition file cannot be read. The script definition might be corrupted, missing, or in an unexpected location.

User response: Make sure that a .rftdef file is in the *workspace*/project/resources directory and that the file is not corrupted. Open the .rftdef file, which is an XML file, to verify that it is not empty and that the file includes correct starting and closing tags. Then try to play back the test again.

CRFCN0613E

The object map cannot be opened.

Explanation: The integer values specified for the *property_name* property in the *object_class_name* object class must be in the range of 0 to 100. The specified value is not within that range.

User response: Change the ObjRecProp.rftop file to set the correct value for the recognition attribute and try to open the object map again.

CRFCN0617E

Playback cannot continue.

Explanation: An attempt was made to set the *property_name* property from a script. The *property_name* property is an object mapping property, which cannot be set from a script.

User response: Verify that object map properties are not set from scripts.

CRFCN0621E

The Flex application cannot be recorded.

System action: When the recorder is about to start recording a Flex application, the recorder was unable to find the player ID. The player ID is required for getting the information about the application.

User response: Ensure that the application is configured and enabled for recording and start recording again.

CRFCN0622E

The script helper class cannot be created

Explanation: An exception is displayed while creating the script helper class. The recorder generates a ScriptHelper class as part of the Rational® Functional TesterHCL OneTest™ UI script. The script helper class creation fails with an exception.

User response: Try recording the script again.

CRFCN0623E

Test objects passed to the application under test must be registered references.

Explanation: The test objects that were passed to the application were not registered references. Test objects that are passed to the application under test must be registered references.

System action: The computer generates a generic Java object for another process (using spy memory). This Java object is a test object and it contains an object reference, which is not a remote proxy reference.

User response: Change the test script so that the test objects that are passed to the application under test have registered proxy references.

CRFCN0624E

Invalid POSIX character class syntax

Explanation: During recording or playback there are instances where regular expressions are evaluated and used. An error is displayed when there is an invalid POSIX character class syntax in the regular expression. Compilation of a character class which represents a regular expression has invalid syntax

User response: Correct the regular expression so that it conforms to the POSIX character class syntax and try recording and playing back the script again.

CRFCN0625E

The template cannot be loaded.

Explanation: The template cannot be located from the specified path.

System action: There are several template files, for example, ScriptSuperHelper template file that help in creating the corresponding types of files. An error is displayed when the template files are not found.

User response: Ensure that Rational® Functional TesterHCL OneTest™ UI is installed correctly and the template files are present at the specified location.

CRFCN0626E

Playback cannot continue.

Explanation: An object map cannot be found. The loadFromStream(java.io.InputStream *input*) method was called on an ObjectMap class, but the input stream input is a null value. An object map is required for this operation, so the input stream cannot be null.

User response: Provide a value for the input stream when you call the loadFromStream(java.io.InputStream *input*) method on an ObjectMap class, and try the palyback again.

CRFCN0628E

The recording cannot be completed.

Explanation: An error is displayed while setting the display location for a dialog box next to its parent. While recording a script, you click a button on the dialog box, a new dialog box opens, the location of this new dialog box is calculated

and is set relative to the parent dialog box. During this operation of calculating the location and positioning the dialog in the user interface an exception is displayed.

System action: The dialog box location cannot be determined.

User response: Adjust the size of the parent dialog box and try the operation again.

CRFCN0639E

The application to be tested cannot be started.

Explanation: An application must be configured correctly so that the Start Application wizard can start the application. The Start Application wizard was unable to start the application. The application might not be configured correctly.

User response: Make sure that the application to start is configured correctly by using the configure application for testing function. For information about configuring applications for testing, see Configuring applications for testing on page 796.

CRFCN0647E

The *script_name* script cannot be played back.

Explanation: An unhandled exception occurred while playing back the script.

User response: Check the playback log for details about the cause of the problem. Correct the issues that are logged, and try to play back the script again.

CRFCN0648E

Playback cannot be completed.

Explanation: During playback channels provide a means of running code that requires GUI thread affinity. If the corresponding GUI thread is unresponsive the channel is also unresponsive. The *request_name* request that must be run with the channel is ignored. The GUI thread and the channel must be responsive to requests.

User response: Close and reopen the application under test. Make sure that the current playback process is closed completely. Then play back the script again.

CRFCN0649W

Playback cannot continue.

Explanation: A null baseline object was passed to the performTest() method call on a test object. A non-null baseline verification point object must be specified to the TestObject.performTest method.

User response: Make sure that the baseline object passed to the method is not null. The baseline is the value being passed to the performTest() method as an argument. You can make sure that the object that is passed to the performTest() method is not null. For example, a performTest() method can be called manually and the method call

directly and indirectly passes a null. This situation can be avoided by adding a null check condition before making a method call.

CRFCN0653I

An empty <obj> tag was encountered.

Explanation: The ObjRecProp.rftop file has an empty <obj> tag. The element for Obj: .RecognitionProperties is empty. The <obj> tags in this file cannot be empty.

User response: Make sure that there are no empty <obj> tags in ObjRecProp.rftop file, and start Rational® Functional TesterHCL OneTest™ UI again.

CRFCN0660E

The edited *script_name* script cannot be saved.

Explanation: The version of the script file saved in the file system and the file displayed in the script editor might not be in sync. To save edited scripts; the file-system version and the version in the editor must be in sync.

User response: Refresh the project containing the script and save it. If you continue to get an error, try recording the script again.

CRFCN0661W

The recognition score of the found object does not qualify the object as a match

Explanation: Playback expects to find the *object_name* object, but the recognition score of the found object does not qualify the object as a match. The nearest match is the *object_found* object with a score of *score_value*. For example; if the playback is trying to locate test object [{0}], the score of nearest match is [{1}].

User response: Verify that the application under test is functioning correctly. Also determine whether multiple instances of the application under test are running. If so, close all instances and try playback again.

CRFCN0663E

The Application View cannot display the *object_name* test object.

Explanation: After recording a script with simplified scripting enabled, the 'Application View' displays the test objects visually. If the test object is not present in the object map then the above error is displayed.

User response: Try recording the script again in a stable Rational® Functional TesterHCL OneTest™ UI environment.

CRFCN0672E

Playback cannot continue.

Explanation: During the construction of the *scriptFullName* script, a *java.lang.Throwable* class was caught. The *scriptFullName* is called by the callScript(String *scriptFullName*, Object[] *args*, int *iterationCount*) method, encounters the error, and cannot continue.

User response: Verify that the script that is specified by the *scriptFullName* value exists in the project location to avoid the problem. If the problem persists, you can omit this line from test script.

CRFCN0673E

The script cannot be played back.

Explanation: An attempt was made to use the RationalTestScript.setScriptName() method to change the name of the script dynamically. Dynamic script changes are only supported for internal functions. If you use this method in a script, the script cannot be played back.

User response: Edit the script to remove the method call, and then play back the script again.

CRFCN0682E

Remote playback cannot continue without dynamic download support.

Explanation: Remote playback is not supported without dynamic download support by the remote server for TPTP-based remote execution. For TPTP-based remote execution, the remote server is requested to download files. If downloading files is not supported by the remote server, then playback is also not supported by the server.

User response: Enable download support for the remote Java sever to support playback of functional test scripts, and try the remote playback again. For more information see **Setting up IBM Rational Agent Controller for Functional Test execution from ClearQuest Test Manager scripts** topic in the information centre.

CRFCN0698E

The recording cannot be completed.

Explanation: The utility class is responsible for creating an empty library class from the default template while recording a script. This problem occurs if the template files from which the helper class is created are not available for the utility class.

User response: A number of factors might have caused this problem. Review the exception message for details about the issues that interrupted this recording. Ensure that the functional testing project is created correctly. Verify that the template file exists in the correct folder. An example of a template file name and location follows: `\templates \ft_scripthelper.java.rfttpl`.

CRFCN0699E

Flags must be a combination of PRE_DOWN, PRE_UP and POST_UP.

Explanation: Certain mouse events such as PRE_DOWN, PRE_UP and POST_UP are ignored by the recorder because the recorder is configured so that it does not call the test object proxies for those events. The filter that prevents

recording these mouse events is not working correctly. Flags must be a combination of PRE_DOWN, PRE_UP and POST_UP. An exception is displayed when the filter does not work correctly.

User response: This filter is configured by default. If the filter does not working correct, there is a problem with the product. Uninstall the testing product, and install it again.

CRFCN0707E

Script playback cannot be started.

Explanation: If the **rational_ft_impl** class is not initialized then the value of datastore object is null. In this case any other operations cannot be performed with this class.

User response: Ensure that the **rational_ft_impl** class is initialized before performing any operations with it.

CRFCN0710E

The application cannot be dynamically enabled. Recording the script cannot continue.

System action: The Rational® Functional TesterHCL OneTest™ UI recorder was unable to dynamically enable the application while recording an action on a running application. To record a script, Rational® Functional TesterHCL OneTest™ UI must be able to enable the application dynamically

User response: Before trying to record the script again, ensure that you configure the application for testing using Rational® Functional TesterHCL OneTest™ UI configuration wizard. For more information see **Configuring applications for testing** topic in the information centre.

CRFCN0712E

The recording cannot be completed as the script helper super class fails.

Explanation: The script helper superclass creation fails while recording a script. The empty script helper superclass is used to insert shared methods into it. If the template files from which the help superclass is created are unavailable, the recording stops. The template files must be available in the correct location during recording.

User response: Make sure that the Rational® Functional TesterHCL OneTest™ UI project is created correctly. Verify that the template file are correctly located. An example of the file name and location follows: `\templates \ft_scripthelpersuper.java.rfttpl`

CRFCN0715E

The application cannot be started.

Explanation: Unsupported attribute types cannot be used for an XML element in a XML File. The *XML_element_tag* tag is an unsupported attribute type. Supported attribute types are as follows: L, N , Z , B , C , S , I , J, F , D , U, Type, T, U8, U32, U64, Decimal

User response: Revise the XML file using supported attribute types for XML elements. Start Rational® Functional TesterHCL OneTest™ UI again.

CRFCN0720E

Error creating the test object map. Playing back the *script_name* script cannot continue.

Explanation: To play back a script, a test object map must be created. The test object map cannot be created because the object map file for the script cannot be found.

System action: An error message is displayed while creating a test object map as storing an object map file on a file system fails.

User response: Ensure that the object map file is stored in the Rational® Functional TesterHCL OneTest™ UI project. For more information see **Creating a new test object map** topic in the information center.

CRFCN0722E

The *script_name* script could not be created.

Explanation: An attempt was made to create a script while the recording function was active. Scripts cannot be created while the recording function is active.

User response: Make sure to stop recording before attempting to create a script.

CRFCN0723E

The script cannot be played back.

Explanation: The script start and end calls are mismatched. If the size of the script end point is zero, playback fails. The LogAdapter writes the test execution results to log during playback. At the script starting point the result size is zero. Toward the script end point, the result size is equal to the number of statements in the script. If the LogAdapter finds that the script end point result size is zero an error is displayed.

User response: Ensure that the size of the script towards the end of the execution is not zero. If the check continues to fail then try executing the script again in a stable Rational® Functional TesterHCL OneTest™ UI environment. If the error persists then it is a product defect.

CRFCN0724E

Playback found *object_value* matching candidate for an test object.

Explanation: During playback, the recorder finds a candidate that matches the search criteria starting at ProxyTestObject. If the more than one candidate matches the search criteria then an error is displayed.

User response: Ensure that no other instance of the application under test is running during playback.

CRFCN0726E

XML: Cannot reconstruct object from data

Explanation: The XML code cannot reconstruct an object from the data. The code expected *class_type1* class but encountered *class_type2* class.

System action: Data that persists in the XML format in the location `<Product installation directory>\FunctionalTester\bin\ObjRecProp.rftop` is reconstructed as objects. During this process, if there is a mismatch between argument types passed to the constructor of the class and the expected type an exception is displayed, the object cannot be constructed.

User response: Ensure that the XML file is not corrupt and try the operation again.

CRFCN0726E

XML: Cannot reconstruct object from data

Explanation: The XML code cannot reconstruct an object from the data. The code expected *class_type1* class but encountered *class_type2* class.

System action: Data that persists in the XML format in the location `<Product installation directory>\FunctionalTester\bin\ObjRecProp.rftop` is reconstructed as objects. During this process, if there is a mismatch between argument types passed to the constructor of the class and the expected type an exception is displayed, the object cannot be constructed.

User response: Ensure that the XML file is not corrupt and try the operation again.

CRFCN0733E

Invalid POSIX character class *class_name*. Playback or recording cannot be completed.

Explanation: The *class_name* Portable Operating System Interface (POSIX) character is not valid. While recording or playback there are instances where the verification point text is evaluated as regular expression. If the verification point contains a regular expression with incorrect syntax, the process cannot continue.

User response: Open the verification point from the script explorer and verify the syntax of the regular expression.

CRFN0752E

The default value of *option_value* is not in the set of valid values.

Explanation: The default value of *option_value* is not in the set of valid values. During playback, when a client test context is created, all option definitions that are loaded from customization files are initialized with default values. These values must be within the set of valid values for the option.

User response: The set of legal default values are present in the customization file located at `<project folder>\templates\ft_scripthelpersuper.java.rfttpl`. Change the default value to a valid value.

CRFCN0754E

The *file_name* keyword file cannot be updated.

Explanation: The keyword file cannot be found or cannot be read. The keyword file must be in the location that is specified. The product must be able to read the file.

User response: Refresh the keyword view and make sure that keyword file in the correct location and can be read.

CRFCN0759E

The script cannot be played back.

Explanation: Supported log types must be specified for playback logs. The specified playback log is not supported.

User response: Specify a supported log type for the playback log and play back the test again. For information about supported log types, see Functional test logs on page 1497.

CRFCN0763E

The test cannot be played back.

Explanation: The test object ID is not in the *test_object_name* object map. For playback, the generated test object ID must be correct, specified in the object map, and must not be empty.

User response: Verify that the unique, generated ID property tag in the .rftxmap file for the script has not been changed or empty. Then try to play back the test again. If the ID property tag in the .rftxmap file is empty or test cannot be played back, record the test again.

CRFCN0768E

Problems obtaining the location of cascading dialog boxes. Recording cannot continue.

Explanation: The display location of a child dialog box could not be calculated. While recording a script if a series of dialog boxes are displayed, the display location of the child dialog box is calculated relative to the location of the parent dialog box. The recording must be able to calculate this location.

User response: Try recording the script in a stable Rational® Functional TesterHCL OneTest™ UI environment.

CRFCN0770E

You cannot pass objects by reference to the application under test unless the object referenced comes from the application under test.

Explanation: You cannot pass objects by reference to the application under test until the object referenced is from the application under test.

System action: To pass a generic Java object to another process (through spy memory) the Java object must either be a proxy or must wrapped in a proxy or must be referenced to an object from the application under test. If the object

that is passed is a client test context object and the system is supposed to pass this as reference to another test context then an exception is displayed.

User response: Modify the Rational® Functional TesterHCL OneTest™ UI script so that the test objects which are passed to application under test has registered proxy references. For more information see **Locating a test object in the application** in the information centre.

CRFCN0771E

The recording stopped.

Explanation: A timeout value in an internal method call was reached. The low-level record turns off, pausing and resetting the recording for restarting.

User response: To try to resume the recording, press **Resume**.

CRFCN0775E

The recording could not resume after a pause.

Explanation: An unknown error occurred during recording.

User response: Stopping, and then restarting the recording might resolve the issue. If the problem persists, contact support.

CRFCN0778E

Playback cannot continue.

Explanation: An attempt was made to find the *map_name* object map file in the *data_store* directory, but the file is not in this location.

User response: Make sure that the *map_name* file is in the *data_store* directory, and try the playback again.

CRFCN0779E

The script being recorded cannot be saved.

Explanation: At the end of a recording session, the script definition is saved by checking all the referenced test objects that are present in the associated map. The reference to the *node_name* node is not in the *map_name* map. The test object that is referred to must be in the map.

User response: Ensure that the current functional testing workspace is not locked for use by any session.

CRFNC0783E

Application under test (AUT) is not responding while trying to find the object : {0}.Try playing back again after restarting AUT. If the problem persists, avoid playing back on the above object.

Explanation: When playing back a functional test script the application under test may not respond while trying to find the object. It is possible that the application is running, but while trying to find a specific object, the object hierarchy is not set properly. This causes an infinite loop in the application and the object is not found.

System action: Playback fails to find an object.

User response: If multiple attempts to find the object fail, avoid interacting and playing back on that object.

CRFCN0784E

Application under test (AUT) is not responding while performing the action : {0} on control {1}. Try playing back again after restarting AUT. If the problem persists, avoid performing the above action.

Explanation: When playing back a functional test script, the application under test is unable to perform a specific action on an object. This could be because the specific action makes the application unresponsive.

System action: Playing back an action on an object fails.

User response: If multiple attempts to perform the action fail, avoid performing that specific action on that object

CRFCN0785E

-{0} must be followed by a playback type

Explanation: This message is displayed when execution variables are specified in an invalid format when playback is invoked from a command line.

System action: Playback fails, throwing the invalid command line exception.

User response: Specify the execution variables in the proper (name=value) format separated by a semi-colon(';'), for example, "login=user1;password=samplepasswd"

CRFCN0786E

Only one -{0} allowed in the command line arguments

Explanation: This message is displayed when either the specified execution variables have been specified in an invalid format, or the file that contains the execution variables is missing.

System action: Playback fails, throwing the invalid command line exception

User response:

Do these actions:

1. Specify the execution variables in the proper (name=value) format separated by semi-colons(';'), for example, "login=user1;password=samplepasswd".
2. Make sure that the file containing the execution variables exists and is present.

CRFCN0787W

Warning: The number of test objects created/registered exceeds the number of test objects unregistered for the script. Unregister the test objects that were not unregistered in the script using the unregister(TestObject), unregisterAll(), and the testobject.unregister() APIs.

Explanation: This message is displayed when test objects requested earlier in the current call script have not been unregistered.

System action: Playback log contains a warning message.

User response: Unregister the test objects that were requested earlier, which were not unregistered in the script, using the unregister(TestObject), unregisterAll(), and the testobject.unregister() APIs.

CRFCN0788W

Warning: The number of test objects created/registered exceeds the number of test objects unregistered for the script. Unregister the test objects that were not unregistered in the script using the unregister(TestObject), unregisterAll(), and the testobject.unregister() APIs.

Explanation: This message is displayed when test objects requested earlier at any time in the current call script have not been unregistered.

System action: Playback log contains a warning message.

User response: Unregister the test objects that were requested earlier, which were not unregistered in the script, using the unregister(TestObject), unregisterAll(), and the testobject.unregister() APIs.

CRFCN0791E

Failed to disable browser. Browser location is {0}.

Explanation: This message is displayed on trying to disable the Mozilla Firefox browser when it is open.

System action: The browser is not disabled.

User response: Close the browser and then disable it.

CRFCN0792E

Failed to enable browser. Browser location is {0}.

Explanation: This message is displayed on trying to enable the Mozilla Firefox browser when it is open.

System action: The browser is not enabled.

User response: Close the browser and then enable it.

CRFCN0793E

Wrong option for dynamic find enablement

Explanation: When dynamic find is used through the command-line playback option, and the option is not true or false, this error message is displayed.

System action: Script playback does not start.

User response: Correct the value to either true or false. Any value other than the boolean value is considered invalid.

CRFCN0794E

The Google Chrome browser was not enabled properly for testing.

Explanation: The Google Chrome browser was not enabled properly for functional testing.

System action: Recording and playback of scripts that are used to test applications running on Google Chrome browsers fail.

User response: If you suspect that the Google Chrome browser has not been successfully enabled for functional testing, use the diagnostic tool to test the enablement of the Google Chrome browser in the **Enable Environments for Testing** dialog box. The tool offers quick and simple directions to solve the problems it finds. Click the **Test** button to open the Browser Enablement Diagnostic Tool on page 799.

The browser may not have been properly enabled due to any of the following reasons:

- No Sun Java Runtime Environment (JRE) has been associated with the browser, or the associated JRE is not enabled. To resolve this, associate Sun JRE 1.6 Update 10 with the Google Chrome browser, and enable the JRE. For instructions to enable the JRE, see Enabling Java environments on page 778.
- The default port specified for the web server used for Google Chrome testing is being used by another application on the workstation. Specify a port number that is not in use, in the Webserver Configuration page on page 856 in the Rational® Functional TesterHCL OneTest™ UI **Preferences** dialog box, and in the Options for the extension IBM® Rational® Functional TesterHCL OneTest™ UI for Google Chrome™. For instructions to do this, see Changing the web server port for communication with Google Chrome on page 789.



Note: Ensure that you specify the same port number in both places. The port number for the web server is used when you enable the browser manually in the **Enable Environments for Testing** dialog box. Enablement fails if the port number has not been specified at both locations.

CRFCP0001E

Proxy format error. Expected the element *DOMAIN_IMPLEMENTATION*. XML contains the element *element2*.

Explanation: The input object from which the named data is accessed is not in the correct format. The proxy format was not specified correctly. The *DOMAIN_IMPLEMENTATION*.xml element that contains the *element2* element was

expected. PersistIn() and PersistOut() are used to write and read data in SPY memory. The exception message is displayed when the rftcust file has been modified for custom controls and there is error in the XML modification.

System action: The PersistIn() operation fails.

User response: Specify the element according to the correct format, and try the operation again.

CRFCP0002E

Invalid Type: *testDataType*

Explanation: The control used for getTestDataTypes() has been implemented but getTestData() has not been implemented. getTestDataTypes() is used to view hashtable of type and description pairs that is used to describe the verification data available from this proxy.

User response: Implement getTestData() for that control. Specify a valid type and try the operation again.

CRFCP0003E

The field *field_name* does not exist, or the caller does not have sufficient permission to access the field.

Explanation: An attempt was made to obtain the *field_name* field. Either the field does not exist or permission requirements prohibit access to the field.

User response: Verify that the field exists and check the permission, and try to run the script again.

CRFCP0004E

You cannot set a property on the Process domain.

Explanation: A setproperty method has been invoked on a DomainTestObject domain. The DomainTestObject domain provides access to the domain.

System action: Playback stops.

User response: Do not call setproperty on DomaintestObject.

CRFCP0005E

Screen rectangle could not be computed.

Explanation: The screen relative rectangle for the user interface control that the test is attempting to act on could not be found. The rectangle is required to perform operations on the control or subitem.

System action: Playback stops.

User response: Verify that the control and the subitem exist and are visible on the screen.

CRFCP0006E

Screen rectangle could not be computed after scrolling.

Explanation: The screen that is relative to the rectangle for the user interface control could not be found. The rectangle is required to perform operations on the control or subitem. Rational® Functional TesterHCL OneTest™ UI cannot scroll to find the object if the object is not visible on the screen.

System action: Playback stops.

User response: Verify that the control and the subitem exist and are visible.

CRFCP0007E

Screen rectangle could not be computed

Explanation: The rectangle object is null or not valid.

System action: The rectangle object cannot be obtained and playback stops.

User response: Contact support.

CRFCP0008E

Screen rectangle could not be computed

Explanation: The rectangle object is null or not valid.

System action: The rectangle object cannot be obtained and playback stops.

User response: Contact support.

CRFCP0009E

Unabled to raise child context menu: *subitem*

Explanation: Playback requires the *subitem* child context menu, the context cannot be opened because the parent menu is null.

System action: The screen rectangle cannot be obtained and playback stops.

User response: Contact support.

CRFCP0010E

Invalid state for action: *state*

Explanation: The state of the check box is indeterminate. The clicktostate operation to click the check box did not change the state of the check box. The *state* check box state must be defined for playback to continue. The state of the check box is not `SELECTED` or `NOT_SELECTED` or if its a radio button,, the state is not selected.

User response: Specify a valid state and run the script again. Set the state using the setState() function.

CRFCP**0011**E

Can not deselect a radiobutton.

Explanation: An attempt was made to clear a radio button selection. However, a radio button cannot be cleared; as the state is already deselected.

System action: The attempt to clear the radio button state cannot be performed and playback stops.

User response: Verify that the radio button is in a selected state before attempting to set the state, and then run the script again.

CRFCP**0012**E

Can not set a checkbox to an indeterminate state.

Explanation: A check box has been set to indeterminate. Check boxes can only be selected or cleared.

System action: Playback stops. An UnsupportedAction exception message is written in the log.

User response: Verify that the state of check boxes is either cleared or selected.

CRFCP**0013**E

AWT List: Unable to scroll into view: *subitem*.

Explanation: The index is out of the visible range. The *subitem* object cannot be scrolled into view. For playback to continue the object must become available on the screen.

System action: The screen cannot be obtained and playback stops.

User response: Try using ensureObjectIsVisible() before performing any action on the control. If the issue persists, contact support.

CRFCP**0014**E

Unsupported action for list control: Item is *item* and Action is *action*

Explanation: An unsupported action was attempted on an item. An *action_name* action was attempted on an *item_name* item. Only select, clear, and extend selection are valid actions. The action is neither of Select, Deselect or ExtendSelect states.

System action: The state is not set, and playback stops.

User response: Specify a valid action and try running the script again. Use the setState(action, Subitem item) function to set the action.

CRFCP0015E

Unsupported action for list control: Action is *action*

Explanation: The *action_name* action is not supported. DeselectAll is the only valid state to set.

System action: The state cannot be set and playback stops.

User response: Specify valid state, and try playing back the script again. Specify the state using setState(Action action).

CRFCP0016E

java.awt.Choice: popup window is not visible, so subitem can not be located.

Explanation: A combination-box popup window is not visible, therefore the subitem in the combination box cannot be selected.

System action: Playback stops. An UnableToFindSubitem exception message is written in the log.

User response: Verify that the combination-box popup window is visible.

CRFCP0017E

java.awt.Choice: can not raise the popup window.

Explanation: A popup window can take time to be populated with the specified subitem. Rational® Functional TesterHCL OneTest™ UI internally waits if the subitem is not visible for a specified amount of time. The subitem is not found even after the wait.

System action: Playback stops. An UnsupportActionException message is written in the log.

User response: Contact support.

CRFCP0018E

setState method for Choice control can only be used to select a subitem. Current action is *action*

Explanation: The *action_name* action is not valid for this control. Only select and scroll actions can be performed. The setState method for choice control can only be used to select a subitem.

System action: The state cannot be set and playback stops.

User response: Specify select or scroll for the choice control and try playing back the script again.

CRFCP0019E

Choice control can only set state to one item at a time: First item is *item1*, last item is *itemlast*

System action: There was an attempt to use the choice control to set the state of more than one item at the same time. The choice control can only be set to one state.

System action: Playback stops.

User response: Specify the action on only one item only and try playing back the script again. Specify the state using setState(Action action, Subitem item).

CRFCP0020E

AWT FileDialog associated window could not be found.

Explanation: The FileDialog window has taken too long to open. The window that is associated with the Abstract Window Toolkit (AWT) FileDialog window could not be found. Playback requires that the window open within a certain time limit.

System action: Playback stops.

User response: Verify that the window that is associated with the AWT FileDialog exists and try playing back the script again.

CRFCP0021E

Unable to activate file dialog - dialog not found.

Explanation: There has been an attempt to activate a file dialog window, but the window cannot be found. The window might already be active because another modal window is open or the required window cannot be displayed.

System action: Playback stops.

User response: Verify that the file dialog window exists and try playing back the script again.

CRFCP0022E

Unable to raise child menu: *parent_object_name*

Explanation: A parent menu is null or not valid, preventing the *parent_object_name* child menu from opening. Playback requires that the child menu open.

User response: Contact support.

CRFCP0023E

Unable to input keys - top level window not found.

Explanation: Keyboard input is performed on a top-level window, but the top-level window is not active.

System action: Playback stops.

User response: Verify that the top level window is active.

CRFCP0024E

Unable to activate window - top level window not found.

Explanation: A top-level window is not found and the action on controls requires that the top-level window is available and active.

System action: Playback stops. A WindowActivateFailedException message is written in the log.

User response: Verify that the top-level window exists before performing any action.

CRFCP0025E

Unable to activate window - top level window not found

Explanation: There has been an attempt to activate a window, but the window cannot be found. The window might already be active because another modal window is open or the required window cannot be displayed.

System action: Playback stops.

User response: Verify that the window exists and can be displayed, and try playing back the script again.

CRFCP0026E

Can not resize a window to a negative size.

Explanation: Windows cannot be resized with negative parameters in the resize() method.

System action: Playback stops.

User response: Specify positive parameters in the resize() method.

CRFCP0027E

Can not move a window to a null location.

Explanation: TA null point is specified as an argument in the resize() method.

System action: Playback stops.

User response: Specify a non-null point as a parameter in the resize() method.

CRFCP0028E

Applet host window is not accessible.

System action: Playback stops. An UnableToPerformActionException message is written in the log.

User response: Ensure that the applet window is visible before performing any action.

CRFCP0029E

This action must be performed against the applet host window

Explanation: The action was not performed against the host applet window. This action must be performed against the applet host window.

User response: Perform the action such as close, resize, maximize, minimize, restore, move, or contextHelp against a host window and try playing back the script again. Use getAppletHostWindow(). action name.

CRFCP0030E

Invalid state for action: *action*

Explanation: The *action_name* action cannot be performed when the application is in the current state. The action must be performed when the application is in the correct state. The control is JRadioButton and the state is not `SELECTED`.

User response: Change to a valid state using mouse modifiers and try playing back the script again. Use setState() function.

CRFCP0031E

JRadioButton does not support setState*state*

Explanation: The state provided is not a valid JRadioButton. JRadioButton does not support the *state_name* setState. The valid state for the JRadioButton is `SELECTED`.

User response: Specify a valid state for JRadioButton using setState(SELECTED) and try to playing back the script again.

CRFCP0032E

Can not set the state to INDETERMINATE

Explanation: The *control_name* control cannot be set to the indeterminate state. Only the selected and cleared states are valid.

System action: Setting the state fails and playback stops.

User response: Specify a valid state and try playing back the script again.

CRFCP0033E

java.jfc.JTree: multiple selection is not allowed

Explanation: Multiple selections were attempted in a java.jcf.JTree option. JTree options do not support multiple selections.

User response: Make a single selection in the JTree option and try playing back the script again.

CRFCP0034E

java.jfc.JTable: Row Selection is not Allowed

Explanation: An row-selection action was attempted in a java.jfc.JTable. JTables do not support row selection.

User response: Specify a supported action and try playing back the script again.

CRFCP0035E

java.jfc.JTable: Column Selection is not Allowed

Explanation: A column-selection action was attempted in a java.jfc.JTable. JTables do not support column selection.

User response: Specify a supported action and try playing back the script again.

CRFCP0036E

java.jfc.JTable: Cell Selection is not Allowed

Explanation: A cell-selection action was attempted in a java.jfc.JTable. JTables do not support cell selection.

User response: Specify a supported action and try playing back the script again.

CRFCP0037E

Can not resize a window to a negative size: *(dimension1, dimension2)*

Explanation: An attempt was made to make a window size negative with these coordinates: *(dimension1, dimension2)*. Windows sizes cannot be negative.

User response: Specify a positive and valid size and try playing back the script again.

CRFCP0038E

Can not move a window to a null location

Explanation: An attempt was made to move a window to a null location. Windows cannot be moved to null locations.

User response: Specify a non-null location and try playing back the script again.

CRFCP0039E

Minimize is not supported

Explanation: This method attempts to minimize the associated window. The means by which the window is minimized depends on the platform. If a window is already minimized this method has no effect. The actions minimize, resize, tofront and moving to a null point is not supported.

System action: Playback fails.

User response: Specify a supported action and try to play back the script again.

CRFCP**0040**E

Resize is not supported

Explanation: There was an attempt to resize an object. Resizing is not supported. The actions minimize, resize, tofront and moving to a null point is not supported.

User response: Specify a supported action and try to play back the script again.

CRFCP**0041**E

Can not move a window to a null location

Explanation: There was an attempt to move a window to a null location. Windows cannot be moved to null locations.

System action: Playback fails.

User response: Specify a non-null location for window and try to play back the script again.

CRFCP**0042**E

tofront is not supported

Explanation: There was an attempt to bring a window to foreground. This window cannot be brought to the front from the background. The tofront action is not supported. The actions minimize, resize, tofront and moving to a null point is not supported.

User response: Perform a supported action or action and try to play back the script again.

CRFCP**0043**E

Invalid state for action: *state*

Explanation: There was an attempt to set the state for an action. The *state_value* is not valid. If there is no toggle or radio button, the state cannot be set. If there is a radio button but no option is selected or if the state is indeterminate then the state is not valid.

User response: Change the state to a valid state using mouse modifiers and try to play back the script again.

CRFCP**0044**E

setState can only be used to scroll the split pane: *action*

Explanation: There was an attempt to scroll a single pane using the function setState(). The setState action can be used only to scroll a split pane. On split pane the action could be a horizontal or vertical scroll. The valid actions depend on the control. The valid actions are: SINGLE_SELECT, DESELECT, EXTENDSELECT, DESELECTALL, CHECK, UNCHECK, UNDETERMINED, VSCROLL, HSCROLL, SCROLL_PAGEUP, SCROLL_PAGEDOWN, SCROLL_PAGELEFT,

SCROLL_PAGERIGHT, SCROLL_LINEUP, SCROLL_LINEDOWN, SCROLL_LINELEFT, SCROLL_LINERIGHT, EXPAND, EXPAND_AND_SELECT, EXPAND_AND_EXTENDSELECT, COLLAPSE, COLLAPSE_AND_SELECT, COLLAPSE_AND_EXTENDSELECT, ACTION_FIRST, and ACTION_LAST.

User response: Scroll a split pane or correct the action for a single pane and try to play back the script again.

CRFCP**0045**E

No screen visible point found

Explanation: The specified coordinates are not visible on the screen. One of the two coordinates is null.

User response: Specify non-null coordinate and try playing back the script again.

CRFCP**0046**E

java.swt.Combo: setState can only be used to select a subitem: *action}*

Explanation: In java.swt.Combo, the setState action can only be used to select an *action*subitem.

User response: Specify a supported action and try playing back the script again.

CRFCP**0047**E

java.swt.CTabFolder: setState can only be used to select a subitem: *subitem_action*

Explanation: In java.swt.CTabFolder, the setState action can only be used to select an *action* subitem.

System action: Playback fails.

User response: Perform only supported operations.

CRFCP**0048**E

Property *name* was not found

User response: Verify the location of the property. Set the value for a valid property and try playing back the script again.

CRFCP**0049**E

Property *name* is read only

Explanation: The *name* property is read-only. The value of a read-only property cannot be set.

User response: Cannot set value to a read only property. Remove the setProperty statement and playback the script.

CRFCP**0050**E

No screen point found for object.

Explanation: While a script was playing back on a control, either the screen point for the control cannot be found or the control is not visible. This error might also occur when dual monitors are used.

System action: Playback stops.

User response: If the control is visible but playback still stops, try a point that is relative to the geometry to the control and play back the script again.

Note: Rational® Functional TesterHCL OneTest™ UI does not support dual monitor issues currently.

CRFCP0051E

You cannot set a property on an Html Domain object.

Explanation: There was an attempt to set a property on an HTML domain object. Properties cannot be set on HTML domain objects.

User response: Remove the property-set action and try playing back the script again.

CRFCP0052E

No screen visible point found for document.

Explanation: During playback Rational® Functional TesterHCL OneTest™ UI has to perform multiple clicks for a single action. After the first click(), if the plugin is not properly enabled, Rational® Functional TesterHCL OneTest™ UI will not be able to record or playback the actions. After a click action, the application cannot return to the correct state.

System action: Playback stops.

User response: Contact support.

CRFCP0053E

Unable to click to desired state: *action*

Explanation:

The *action* state cannot be set or selected. The state is either indeterminate or no radio button has been selected. When you click to change the state of the control, the control might be in one of these conditions:

- A check box and the state are indeterminate.
- A radio button is set but the radio button is not selected.

System action: Playback fails.

User response: Use setState() to set the state of the control. For radio button, the valid state is selected. For check box, the valid states are selected or not selected.

CRFCP0054E

Unable to set to desired state.

Explanation: A check box or radio button has been set to indeterminate state programatically.

System action: Playback stops.

User response: Check if the state of the check box or radio buttons is supported by the control. If the problem persists, contact support.

CRFCP0055E

Unable to set text in a read-only text area

Explanation: There was an attempt to set text in an area that is read-only. Text cannot be set in a read-only area.

User response: Remove the text-set action and try running the script again.

CRFCP0056E

No screen visible point found for form.

Explanation: During playback Rational® Functional TesterHCL OneTest™ UI has to perform multiple clicks for a single action. After the first cick(), if the plugin is not properly enabled, Rational® Functional TesterHCL OneTest™ UI will not be able to record or playback the actions. After a click action, the application does not return to the correct state.

System action: Playback stops.

User response: Contact support.

CRFCP0057E

No Subitems, Image is not an Image Map.

Explanation: This error message is displayed when Rational® Functional TesterHCL OneTest™ UI gets the subobject that matches the subitem specification for an image control. The image might not have the *isMap* or *useMap* DOM properties to specify that it is an image control.

System action: Playback stops and the message is displayed in Rational® Functional TesterHCL OneTest™ UI console and in playback log.

User response: Verify that the image has the *isMap* or *useMap* DOM properties.

CRFCP0058E

Can not resize a window to a negative size: *({dimension1}, {dimension2})*

Explanation: There was an attempt to set the window to a negative size with these dimensions: *dimension_1* and *dimension_2*. A window cannot be set to a negative size.

User response: Specify a positive size for the window, and try playing back the script again.

CRFCP0059E

Can not move a window to a null location

Explanation: There was an attempt to move a window to a null location. Windows cannot be moved to null locations.

User response: Specify a non-null location for the window and try playing back the script again.

CRFCP0065E

Playback cannot continue.

Explanation: A top-level object was not found for a control. Top-level objects for controls must be available. Top-level objects have no parents. During playback all the top-level objects are obtained from the application and then the control where the GUI action is to be performed is searched inside a top-level Object. For example, Form is a top-level object. The application under test can be designed such that a top-level object is in foreground when a graphical action is performed on it. If during playback the top-level object is not in foreground, the scripted action cannot be performed.

User response: Verify that objects on which actions are made in the script are visible during playback.

CRFCP0067E

Playback cannot continue.

Explanation: The DescribedObjectReference class was unable to interact with a top-level window. Before a script can interact with a window, the window must be brought to the foreground. Sometimes, however, the target window cannot be brought to the foreground. For example, if a modal dialog box, such as an authentication or login window, is open, the script cannot interact with the target window. The main window must be active for interaction.

User response: Design the application so that the top-level application window is active and interacted with.

CRFCP0068E

The MXML file cannot be compiled into a .swf file.

Explanation: An error message is displayed while compiling mxml to swf at the command prompt.

System action: During compilation, the MXML file is compiled with Flex automation libraries and Rational® Functional TesterHCL OneTest™ UI Flex libraries. The corresponding .swf file is generated and is tested by Rational® Functional TesterHCL OneTest™ UI. With the .swf file, a batch file, and an HTML page is also generated that corresponds to the MXML file. When compilation fails, the JRE and, the .swf file cannot be generated.

User response: Ensure that the JAVA_HOME variable is set. Run the batch file generated during compilation at the Adobe® command prompt to generate the .swf file and use it for testing purpose. You can run the batch file at the

Microsoft® Windows® command prompt to get the details of the error that occurred and take appropriate action based on that information.

CRFQM0001E

The adapter is unable to connect to the project_name project area in IBM Rational Quality Manager.

Explanation: The Rational® Functional TesterHCL OneTest™ UIand Rational Quality Manager integration adapter fails to connect to the Rational Quality Manager project area.

System action: The Rational® Functional TesterHCL OneTest™ UI and Rational Quality Manager integration adapter connects to a default project instead of connecting to a project area on the Rational Quality Manager server.

User response: Try connecting the Rational® Functional TesterHCL OneTest™ UI and IBM® Rational® Quality Manager integration adapter to the IBM® Rational® Quality Manager project area again. Verify that all steps are complete and settings are correct.

CRFQM0002E

The integration adapter was unable to connect to the server.

Explanation: If the Rational® Quality Manager server is not running and available, the integration adapter for Rational® Functional TesterHCL OneTest™ UI cannot connect to the Rational® Quality Manager server.

System action: The Rational® Quality Manager adapter for Rational® Functional TesterHCL OneTest™ UI does not retrieve project area information from the Rational® Quality Manager server does not disconnect correctly.

User response: Ensure that the Rational® Quality Manager server is running and available and the Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter is connected to the server.

CRFQM0003E

The script could not be run: The project cannot be found.

System action: Rational® Quality Manager received a request to run a script as Rational® Quality Manager connects with the Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter. The adapter cannot run the script because the Rational® Functional TesterHCL OneTest™ UI project to which the script belongs cannot be found.

User response: Verify the location of the Rational® Functional TesterHCL OneTest™ UI project. Ensure that Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter is specifying the correct location.

CRFQM0004E

The script name cannot be run

Explanation: Rational® Quality Manager has connected with a Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter and received a request to run a script.Rational® Functional TesterHCL OneTest™ UI cannot run the test script because the scripts cannot be found in the specified location.

User response: Ensure that the Rational® Functional TesterHCL OneTest™ UI script can be found at the location that is specified for the Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter.

CRFQM0005E

The script cannot be run as the script is not built.

System action: Rational® Quality Manager is connected with a Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter. Rational® Quality Manager receives a execution request for a script. The adapter cannot run the script as the script is not compiled or built.

User response: Ensure that Rational® Functional TesterHCL OneTest™ UI script exists and it is complied or built at the location where the Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter is trying to find it.

CRFQM0006E

The script {0} cannot be executed.

Explanation: Rational® Quality Manager adapter for Rational® Functional TesterHCL OneTest™ UI cannot run the script

System action: Rational® Quality Manager is connected with a Rational® Quality Manager - Rational® Functional TesterHCL OneTest™ UI adapter. Rational® Quality Manager receives a execution request for a script. The adapter cannot run the script

User response: Refer the Rational® Functional TesterHCL OneTest™ UI logs.

CRFQM0007E

Failed to make task data available to execution.

Explanation: Rational® Quality Manager adapter for Rational® Functional TesterHCL OneTest™ UI creates a temporary task file for the request it receives. This error occurs if the creation of the task file fails.

System action: Adapter either fails to create a temporary task file or fails to write the task data to the temporary file.

User response: Ensure that temporary file creation is permitted on the computer where the adapter is running.

CRFQM0008W

Server URL must be in the form {0}. Are you sure {1} is correct?

Explanation: The server URL specified on the Rational Quality Manager adapter user interface is not in the correct form.

System action: A Warning message box is displayed with the message 'Server URL is in the form https://<server>[:portnumber]/<ContextRoot>'

User response: Specify the server URL in the correct format.

CRFQM0009E

You cannot start multiple adapters with the same configuration on the same workstation.

Explanation: An adapter instance is trying to connect to the server while another adapter instance on the same workstation has already connected to the same server configuration using the same details. Multiple adapters with the same configuration cannot be started on the same workstation.

System action: This error message is displayed in the adapter console of the second instance of the adapter interface.

User response: Close the second adapter instance.

CRFWW0001E

Could not open file: {0}. Check if file is a part of workspace

Explanation: The file is not a part of the workspace. To be opened, files must be a part of the workspace.

User response: Check whether the file is a part of workspace. If the file is not part of the workspace, import the file to workspace and try the operation again.

CRFWW0002W

Do you want to delete the existing private dataset?

System action: Displays the message and waits for user response.

User response: Decide whether you want delete the private dataset or not.

CRFWW0003E

Problem initializing the IDE.

Explanation: During recording or playing back a script, a reference to the IDE is required. When the IDE is launched and registering the IDE in the shared memory is attempted internally by Rational® Functional TesterHCL OneTest™ UI, the registration operation fails.

System action: The IDE is not registered.

User response: Try the operation again. If the error persists, contact support.

CRFWW0004E

Critical files are corrupted, missing or cannot be loaded. Reboot your system. If problem persists please reinstall Rational® Functional TesterHCL OneTest™ UI.

Explanation: Critical files are corrupted, missing, or cannot be loaded.

User response: Restart the computer. If the problem persists reinstall Rational® Functional TesterHCL OneTest™ UI.

CRFWW0006E

Problem enabling default environments for testing.

Explanation: In Rational® Functional TesterHCL OneTest™ UI, default JRE or default browser could not be enabled.

System action: Playback cannot be started.

User response:

To resolve this problem, try one of these actions:

- Enable the default JRE and default browser.
- Restart the computer.
- Ensure that the default JRE path and default browser path are correct in the Enable Environment dialog.

CRFWW0010E

The major version of the project is not compatible with the product version.

Explanation: The project version is not compatible with the product version. The might occur when a project that was created using an older version of Rational® Functional TesterHCL OneTest™ UI is used with new version resulting in the project being incompatible with the version of the product. The user is given an option to upgrade the project to make the project compatible with the product version.

System action: Project does not load.

User response: Check that the major version of the project in compatible with the product version. Right-click the project and select **Properties**. Select **Functional Test Project** to see the project version.

CRFWW0011E

The minor version of the project does not match the minor version of the product.

Explanation: The project version does not match the version of the product. The might occur when a project that was created using an older version of Rational® Functional TesterHCL OneTest™ UI is used with new version resulting in the project being incompatible with the version of the product. The user is given an option to upgrade the project to make the project compatible with the product version.

System action: Project does not load.

User response: Check that the project version is compatible with the minor version of the product. Right-click the project and select **Properties**. Select **Functional Test Project** to see the project version.

CRFWW0013E

The major version of the project is not compatible with the product version. This project will be disconnected.

Explanation: The project version does not match the version of the product. The might occur when a project that was created using an older version of Rational® Functional TesterHCL OneTest™ UI is used with new version resulting in the project being incompatible with the version of the product. The user is given an option to upgrade the project to make the project compatible with the product version.

System action: Project does not load.

User response: Check that the project version is compatible with the minor version of the product. Right-click the project and select **Properties**. Select **Functional Test Project** to see the project version.

CRFWW0017E

Please wait for another script to finish.

Explanation: A script was already running when an attempt was made to run another script. Two scripts cannot run simultaneously.

User response: Wait for the current script playback to finish before running another script.

CRFWW0018E

Unable to find script.

Explanation: The script has been removed, moved, or the specified path might be incorrect.

System action: The script does not load.

User response: Verify that the path of the script is correct and that the script is present.

CRFWW0019E

Classpath error: {errmsg}

Explanation: A core exception occurred while the default runtime *classpath_name* classpath was being accessed.

User response: Follow the instructions in the error message. If the problem persists contact support.

CRFWW0020E

Try resetting the project java build path to resolve errors encountered running the script.

Explanation: Errors in the build path prevented the script from running.

System action: The script is not run.

User response: Reset the project Java build path to resolve the errors. Select **Project > Properties > Java Build Path**. Verify that all the required libraries (Jars,external, Jars,variable, library, class folder and external class folder) are added so that all the components on which the project is dependent is added. Select the required items under **Order and Export** tab on which the project has dependency. If this error persists, contact IBM Software Support.

CRFWW**0021**E

Couldn't find a script to playback. Please check your selection.

Explanation: No script is active for playback.

System action: Script playback does not start.

User response: Check your selection and try the operation again.

CRFWW**0023**E

The specified dataset iteration count [count] is not valid.

Explanation: The specified dataset iteration count, *[count]*, is not valid. The iteration count value is either negative or greater than the maximum number of iterations possible. The maximum number is the number of rows in the dataset.

User response: Specify a valid iteration count.

CRFWW**0024**E

Object Map does not exist

Explanation: You open an object map that does not exist. Object map file is either deleted, moved, renamed or corrupted

System action: Playback stops and an exception is displayed.

User response: Ensure that the object map exists.

CRFWW**0025**E

Problem renaming script assets

Explanation: The new name of the script assets might not be valid or might already be in use. Valid script names must be unique.

User response: Specify a valid asset name.

CRFWW**0026**E

Problem opening file

Explanation: The correct permissions to open the file might not be set. To open a file the permissions settings must be correct.

User response: Check the file permissions from **File > Properties** and retry the operation.

CRFWW0028E

Problem deleting script assets

Explanation: You cannot delete the script assets. Test assets are either deleted, renamed or corrupted.

System action: The Rational® Functional TesterHCL OneTest™ UI assets cannot be opened.

User response: Verify that the test assets are not being used by another user.

CRFWW0030W

The destination file exists. Do you want to overwrite it?

System action: Waits for user response.

User response: Decide if you want to overwrite the destination file or not.

CRFWW0031E

There are no files to export.

Explanation: This error message is displayed when you try to export the functional test project from a location that has no files to export

System action: No files are exported

User response: No action required.

CRFWW0032E

Problem encountered during the export.

Explanation: This error occurs when you try to export functional test project or dataset and the files are not saved in the destination location

System action: Export operation fails

User response: Find out the reason for the failure of the write operation. Possible reasons: Invalid destination location, not writable, not enough disk space.

CRFWW0033W

Unable to access script assets for *scriptFile*. Do you wish to proceed without this script?

Explanation: The script assets might not exist or might have been erroneously deleted.

System action: Waits for the user response.

User response: Decide whether to continue opening the project without the script or not.

CRFWW0034E

Problem encountered during the import

Explanation: This error occurs when you try to import a functional test project or a dataset

System action: Import operation fails

User response: Check the error message details to find out the problem and resolve it.

CRFWW0035E

Refresh failed.

Explanation: This error occurs when you try to import the functional test project or dataset and there is a problem while refreshing the project after the import.

System action: Refresh operation on the destination project fails

User response: Use F5 key in the project explorer to refresh the project manually. Also check the error message details to find out the problem and resolve it.

CRFWW0036E

Unable to update the destination.

Explanation: The specified destination might not be writable with current permission settings. To export a project, the destination must have permissions set to enable writing.

User response: Verify that you have write privileges in the path specified for export from **File > Properties**. Try the operation again.

CRFWW0037E

Unable to create the destination directory

Explanation: The permissions set for the destination directory do not permit writing. To export a project, the destination directory must have permissions set to enable writing.

System action: The project is not exported.

User response: Verify that you have write privileges in the path specified for export. Try the operation again.

CRFWW0038E

Publish failed on {fileName} The following files were updated.{updatedFileList} Do you want to undo the operation? Select No if you want to continue publishing the rest of the files.

Explanation: Correct permissions might not be set for the file. Correct permissions are required for exporting files.

System action: The application waits for a response.

User response: Decide whether to continue exporting the remaining files or undo the operation. To export the other files, verify that you have write privileges for the files, and then export them to the destination.

CRFWW0039E

Unable to create backup directory

Explanation: The path for the exported files might not have been specified correctly. If the backup directory cannot be created, the export operation cannot continue.

User response: Verify that the path for the backup directory is specified correctly, and try the operation again.

CRFWW0040E

Unable to copy file to backup directory

Explanation: The permissions that are set for the file or the destination directory might not permit the file to be saved to a temporary backup location Without the backup, the import process continue.

User response: Check the permissions of both the file and the backup directory and verify your privileges. Try the operation again.

CRFWW0041E

Unable to read the file *{fileName}* while creating backup

Explanation: The permissions for the file might not be set to enable reading. To create the backup and continue the export operation, the files must be read.

System action: The import fails because the file or files to be exported cannot be read

User response: Ensure that you have read privileges in the path specified for the files to be exported. Try the operation again. If the error persists, contact support.

CRFWW0042E

Problem encountered during the publishing.

Explanation: The permissions for the file might not be set to enable reading. To continue the import operation, the files must be read.

System action: The import operation fails because the files that have to be exported cannot be read.

User response: Ensure that you have read privileges for the files to be exported. If the problem persists, contact support.

CRFWW**0043**W

Unable to access script assets for *scriptFile*. Do you wish to proceed without this script?

Explanation: The script assets might not exist or might have been erroneously deleted.

User response: Decide whether to continue opening the project without the script or not.

CRFWW**0044**W

dataset structure changed. Are you sure you want to change the association?

System action: Waits for user response.

User response: Decide if you want to change the dataset association or not.

CRFWW**0045**E

Problem associating dataset with scripts

Explanation: The dataset cannot be associated with scripts. To run a script, the specified datasets must be associated with the script.

User response: Try to rectify the error based on the error message, else contact support.

CRFWW**0046**E

Could not insert data driven commands

Explanation: The data-driven commands could not be inserted.

User response:

CRFWW**0047**E

Problem encountered during the export.

Explanation: The correct permissions for writing to the destination directory might not be configured. To export files, permission to write to the destination directory is required.

User response: Verify that you have the permission to write to the destination directory from **File > Properties**, and try the operation again.

CRFWW**0048**E

Could not create dataset

Explanation: The correct permissions for writing to the specified path are not configured. To create datasets, permission to write to the specified path is required.

User response: Verify that you have the permission to write to the specified path from **File > Properties**, and try the operation again.

CRFWW0049E

Create dataset problems

Explanation: Although a dataset file was saved, the file has problems.

User response: Refer to the exception message for more information about the problems, and try to correct the error. If the error persists, contact support.

CRFWW0050E

Could not create dataset

Explanation: The correct permissions for writing to the specified path are not configured. To create datasets, permission to write to the specified path is required.

User response: Verify that you have the permission level to write to the specified path, and try the operation again.

CRFWW0051E

Error showing dataset page.

Explanation: An internal errors prevents a dataset from being displayed the script explorer.

System action: The dataset view is not opened.

User response: Try the operation again. If the error persists, contact support.

CRFWW0052W

Scripts associated with this dataset will be made unusable. Are you sure you want to remove dataset association?

Explanation: This is a warning before deleting a script asset.

User response: Decide whether you want to remove the dataset association or not.

CRFWW0053E

Cannot modify a readonly file!

Explanation: The file has a read-only attribute and cannot be modified.

User response: Modify the file permissions from **File > Properties** and try the operation again.

CRFWW0054W

The following scripts are already associated with a dataset. Changing the dataset associated with a script can cause the script to run incorrectly. Do you want to change the dataset associated with each script?

System action: Waits for user response.

User response: Decide if you want to change the dataset associated with each script or not.

CRFWW0055E

Problem associating dataset with scripts

Explanation: The script definition file might be read-only. The script definition file must be configured for changes to associate a dataset with scripts.

System action: The dataset is not associated with scripts.

User response: Verify the permissions for script definition file, and try the operation again.

CRFWW0056E

Problem encountered during the import.

Explanation: Files in this format or of this type cannot be imported. File must be in these formats or of these types to be imported.

User response: Check the file format and file type. Change the format or type, and try the operation again.

CRFWW0057W

dataset *datasetName* has been modified. Save changes?

System action: Waits for user response.

User response: Decide if you want to save changes made to the dataset or not.

CRFWW0059E

Could not create Object Map

System action: The object map is not created.

User response: Contact support.

CRFWW0060E

Could not create script

Explanation: The recording operation was unsuccessful, so the script cannot be created. A script requires a completed recording.

User response: Try the recording operation again. If the recording fails, contact support.

CRFWW0061E

Script copied however an error occurred updating the script with its new name and location. Any compile errors need to be fixed manually.

Explanation: Files that the script requires might not have been copied while the script was being renamed. All required files must be copied to the new location for the script copying operation to be completed.

User response: Verify that all the files that the script requires are copied to the new location. If the error persists, contact support.

CRFWW0062E

Could save new script

System action: Recording stops and an exception is displayed.

User response: Verify that the path is correct and you have appropriate privileges to access the scripts.

CRFWW0063E

Could not save the editor's copy of the script to the new location. Any changes made to the script since the last save have been lost.

System action: Editors copy cannot be saved in the new location.

User response: Verify that you have appropriate privileges to access the folder.

CRFWW0064E

Could not record into script

System action: An Error Dialog box with the error ID and the message is displayed after the recording session closes.

User response: No user response required.

CRFWW0065E

Please make sure that a Functional Test script is active

Explanation: The test script is not active. To run, the test script must be active.

User response: Activate the test script by selecting the script from the project and try the operation again.

CRFWW0066E

Please select the line where callScript should be inserted

Explanation: No line has been selected to insert the callScript function. A line must be selected to insert a callScript function.

User response: Select a line to insert the callScript function and try the operation again.

CRFWW**0068**E

Make sure that you selected one or more scripts or test folders containing scripts and that these scripts are from the same project as current script.

Explanation: A script or test folder that contains scripts from the same project as the active script was not selected for the callScript function. The scripts and test folders that contain scripts must be from the same project as the active script.

User response: Select a script or a test folder that contains scripts from the same project as the active script and try the operation again.

CRFWW**0069**W

Only scripts from the current project have been inserted

User response:

CRFWW**0070**E

Could not insert test object(s)

Explanation: This problem is encountered when you insert a test object from Rational® Functional TesterHCL OneTest™ UI Java IDE.

System action: An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW**0071**E

Could not rebuild project

Explanation: The library files or jar files are missing.

System action: The project cannot be rebuild.

User response: Verify that the required jar files or library files exists.

CRFWW**0073**E

Could not create script helper superclass

Explanation: Rational® Functional TesterHCL OneTest™ UI cannot create script helper class or file.

System action: Recording stops and an exception is displayed.

User response: You must restart Rational® Functional TesterHCL OneTest™ UI and rebuild the script again

CRFWW0074E

Specified project path does not exist

Explanation: The specified project path does not exist.

User response: Create the directories specified in the path or specify another path and try the operation again.

CRFWW0075W

There are two connections "{0}" and "{1}" to the same project. But the associated test script source in current project can only point to one of the copies. You should disconnect from one of them.

Explanation:

System action:

User response:

CRFWW0078E

Error deleting files

System action: An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW0079E

Problems renaming project item(s)

Explanation: Application can encounter an error when you rename the Rational® Functional TesterHCL OneTest™ UI project assets. For example: dataset name, project name, script name.

System action: The project does not get renamed. An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW0080E

Could not open project

Explanation:

This problem is encountered when:

- Opening a project from the Rational® Functional TesterHCL OneTest™ UI Java IDE.
- While updating the datastore definition. For example: association of the existing project with Test Manager.

System action: The project does not open. An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW0081E

Could not create new project

Explanation: New project cannot get created in the Rational® Functional TesterHCL OneTest™ UI Java IDE

System action: A new project does not get created. An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW0082E

Could not open log.

Explanation: The log from the Rational® Functional TesterHCL OneTest™ UI integrated development environment cannot be opened because the editor that is associated with the log could not be found.

System action: The log is not opened.

User response:

To open the log, try these actions:

- Ensure that the default browser is installed properly.
- Verify that the file association for the extension of the log file is valid.

CRFWW0083E

Select a verification point.

Explanation: A verification point has not been selected. To perform the comparison, a verification point must be selected.

System action: The comparator is not started.

User response: Select a verification point and try to run the comparison again.

CRFWW0084E

Could not create test folder

System action: New folder is not created. An Error Dialog box with the error ID and the message is displayed.

User response: No user response required.

CRFWW0085E

The project is locked for publish by other users/application.

Explanation: The project is locked by another user or application. The files for export must be released from all other users or applications.

System action: The project is locked by another process.

User response: Discover and stop the process that is accessing the project files and try the operation again.

CRFWW0094E

The specified dataset could not be created.

System action: The application is unable to create a dataset file with a valid path name. On some computers, creating files requires that the user have specific permissions. Without these permissions, the file cannot be created.

User response: Check user permissions for creating files. Obtain the correct permissions, if necessary. Try to create the dataset again.

CRFWW0095E

The nested text editor could not be created. The simplified script editor does not display the **Java** tab.

Explanation: Eclipse cannot create the **Java** tab in the simplified script editor.

User response: Close the script and try the operation.

CRFWW0096E

The simplified scripts cannot be exported as HTML or XML files.

Explanation: In some instances the correct permissions might not be set for writing to the target or reading the script at the source. Correct permissions are required for reading the file to be exported and writing the exported file in the specified location.

User response: Check file write permissions to the target for the export. Check the read permissions for the simplified script file. Correct permissions, if necessary, and try to export the simplified script again.

CRFWW0097I

The file from a different workspace cannot be opened.

Explanation: The simplified script that is displayed is not a part of a project that is currently displayed in the Rational® Functional TesterHCL OneTest™ UI workspace.

User response: Connect to the project that the script belongs to and then open the required script again.

CRFWW0098E

The copied action cannot be pasted under the current selection. Make sure the target is not a child of the source or the source itself.

Explanation: The paste target might be a child of the source or the source itself. The paste target cannot be a child of the source or the source it self.

System action: The simplified script step is not moved.

User response: Drag the simplified script step to a valid location.

CRFWW0099W

The dataset association cannot be removed.

Explanation: The dataset that is being deleted is used by steps within the group that is associated with the dataset. dataset associations cannot be removed when data driven steps in the associated group use the dataset.

User response: Remove the data driven steps from the group or associate the group with a different dataset. Try to remove the dataset association again.

Chapter 11. Reference Guide

This guide describes the additional topics to gain more knowledge about Rational® Functional TesterHCL OneTest™ UI.

Reference information for the UI Test perspective

In this section, you will learn all the reference or additional information about the Web UI Test perspective.

Mobile test preferences

You can change the Mobile Test preference settings.

Mobile web testing preferences

The Mobile web testing preferences control how the information are displayed in the Mobile test editor and the Mobile data view.

To access the Mobile web testing preferences, click **Window > Preferences**, expand **Test**, and click **Mobile Web testing**. After changing settings, click **Apply**.

The settings apply to the mobile device editor, Mobile applications editor, and the Mobile testing editor. They are used to distinguish important information from plain text.

List title

Applies to the list of devices and applications of available devices in the **Mobile Devices** editor and to the list of available applications in the **Mobile applications** editor. This is the color of the names of Mobile devices and applications.

List property

Applies to the list of devices and applications of available devices in the **Mobile Devices** editor and to the list of available applications in the **Mobile applications** editor. This is the color of the main properties indicated with the title of the devices and applications.

Available property

Applies to the main properties pane in **Mobile Devices** editor and to the application description pane in the **Mobile applications** editor. This is the color of the main properties available for the current device or application.

Error

Applies to the error detected in the Mobile editors.

Active filter input background

This is the color of the filters when they are active.

The settings apply to the Mobile data view that is associated to the **Mobile test** editor.

UI Test Application Editor preference

This setting is applicable when the same integrated development environment is used to develop and test an application. Developers who use the Eclipse integrated environment and the Android Developer Tools (ADT) can use this setting to quickly modify and re-test an application during the development process itself.

Context: The **Keep only the latest version of the build for the currently worked on application version** option is available from the test workbench toolbar (**Window > Preferences > Test > UI Test > UI Test Application editor**).

Keep only the last application for a version

When this preference is selected, an application that is modified and then built by using the **Run as > Test with** options, is instrumented and the latest build of the application is added to the workbench. Thus, by using this setting, you replace any previous build of the same version of the application by the latest builds. The test suites that were created from the previous build of the same version of the application are automatically associated with the new build.

If the preference is not selected, when you run a build for a modified application by using the **Run as > Test with** options, a message is displayed to indicate that test suites referencing other versions of the same application have been found. You must click **Preview** to open a refactoring wizard and click **Finish** to start refactoring. After they are refactored, the test suites will be used with the new version of the application.

Mobile test reference

Read some of the reference topics about testing mobile applications.

Values for device selection variables

You can create a variable using one of the following reserved names:*RTW_Mobile_Device_Properties* or *RTW_Mobile_Selected_Device*. The variable will be used to enable the selection of a device in your tests. In your variable, you must enter strings that include the device's properties and the associated values, and the strings must comply with the syntactic rules detailed in this topic.

Name

In the **Data elements details** area that opens when you create a variable, enter one of the following reserved variable names: `RTW_Mobile_Device_Properties` or `RTW_Mobile_Selected_Device`.

- *RTW_Mobile_Device_Properties*: This variable must contain a valid selection sentence.
- *RTW_Mobile_Selected_Device*: This variable contains, by default, the ID property of the previously selected device for the current virtual user. It can also be set up explicitly by the end-user with the ID property of the selected device.

Initialize to text

To initialize the variable to a specific value, enter one or multiple selection strings in the **Text** field. The strings consists of the following items: `property's name` followed by `operator value` followed by `property's value,`. The strings are separated with commas. Example: type = Android.

Device properties

Table 57. Main device properties

Key	Content (value)
type	Type iOS or Android
description	A string containing the device model and its brand
apilevel	Starting from 8 for Android and 60000 for iOS (60000 is for 6.0, 60100 is for 6.1)
width	Width of the screen in pixels
height	Height of the screen in pixels
locale	Configured locale (language or country code)
simulator	True for an emulator or simulator, false for a real device
landscape	True if the device is landscape oriented, otherwise false (example: portrait oriented)
gps	True if GPS is available and active, otherwise false
phone	True if phone is available, otherwise false
bluetooth	True if Bluetooth is available and active, otherwise false
id	This is a unique identifier generated for a device. It is displayed in the device category of detailed properties of the **Mobile Device** editor. It must be used in the *RTW_Mobile_Selected_Device* variable.

Other properties can be used. They are displayed in the **Mobile Device** editor.

Example: Example of selection string in the variable: `type = Android, apilevel >= 15, description : ABrandName`. This selection string will enable selection of the first Android device with an API level greater than or equal to 15 and whose description contains a brand name.

Syntactic rules

```
sentence: property-expr { , property-expr }*
  property-expr: property-name operator value
```

```
operator: = | != | < | > | <= | >= | : | =* | !=*
value: boolean | decimal-number | word | quoted-string
boolean: true | false
integer: optional-minus-sign [digit]+
floating-number: optional-minus-sign [digit]+.[digit]+
word: [A-Za-z$_][A-Za-z$_0-9]*
quoted-string: 'any-char-1' | "any-char-2"
```

Where:

- `property-expr` must be set to allow a device selection
- `property-name` corresponds to a property listed above in the table of main properties, or to other advanced properties (see the detailed properties section in the **Mobile Device** editor).
- The `=` and `!=` operators are valid for Boolean properties.
- The `=`, `!=`, `<`, `>`, `<=` and `>=` operators are valid for numeric and lexicographical properties. They are not case sensitive.
- The`:` operator is used to check the lexicographical content of the value entered for the device's property in the selection string. The value is case sensitive
- The `=*` and `!=*` operators are used to check whether the value entered for the device's property in the selection string is interpreted as a regular expression.
- Two kinds of quoted strings depend on their enclosing character `'` or `"`. They are used for a string value containing more than one word or containing special characters, such as regular expressions
- `'any-char-1'`: you can enter all kinds of characters in single-quoted literals, and the `'` character must be doubled. Example: It"s "John", which results in: It's 'John'.
- `'any-char-2'`: you can enter all kind of characters after a double-quoted literals, but it must be followed by a backslash. Example: `"weird\"content\\with-special'chars"` which gives: `weird"content\with-special'chars`
- Prefer single-quoted literals for regular expressions because they do not require adding other backslashes.
- The accepted regular expressions are the ones defined in the documentation of Pattern class, Java 6.

UI Test result reports

By default, the Unified Report is displayed after a mobile test is run from devices, simulators, emulators, or workbench. A statistical report is generated if the test is generated from Rational® Functional TesterHCL OneTest™ UI only. The mobile web report gives details on each action recorded and played back including the think time, the response time and warning messages that help you to identify problems. The statistical report summarizes the health of the run, displays the data that is the most significant to the run, summarizes the resource monitoring performance and gives an average step response time graph.

UI Test Statistical report

Overall page

The following report begins with a summary:

Run Summary

Summarizes the information on the test that was run: elapse time, run status, name of the executed test.

UI Test Health

Summarizes the health of the run and displays the data most significant to the run. This data includes the number of application AUT launched during the test execution, the total number of step attempted and completed, the percentage of completion, the number of steps that failed, if any, and the number of errors.

Step Performance

The report contains a line graph that shows the 10 highest average step response times for the run. The table that follows the graph provides the following information for each step:

- The minimum request response time for that step in the run.
- The average request response time for the run.
- The maximum response time for that step in the run.
- The standard deviation response time for the run.
- The rate for attempts.
- The number of attempts.

You can sort the order of the test steps, which are listed in the **Step** pane, either by alphabetical order or order of execution of the test steps. The default sorting of the test steps is by order of execution.

You can either click the **Execution order** icon or the **Alphabetical order** icon to toggle between the sorting options. You can also click the Up Arrow icon to sort the test steps in the correct order of execution or the correct alphabetical order. Similarly, you can click the Down Arrow icon to sort the test steps in the reverse order of execution or the reverse alphabetical order.

Reference information for the Functional Test perspective

In this section, you will learn all the reference or additional information about the Functional Test perspective.

Test application domain support

Rational® Functional TesterHCL OneTest™ UI is an object-oriented automated testing tool that tests Windows®, .NET, Java™, HTML, SAP, AJAX, PowerBuilder, Flex, Dojo, Visual Basic and GEF applications.

You can also test Adobe® PDF documents, and zSeries®, iSeries®, and pSeries® applications.

Rational® Functional TesterHCL OneTest™ UI also supports Oracle Forms through the use of a proxy.

Adobe PDF documents support

IBM® Rational® Functional TesterHCL OneTest™ UI supports testing of Portable Document Format (PDF) read-only files created for Adobe Reader versions 7.0, 8.0, 9.0, 10.0, and 11.0. (Adobe Reader 11 is the last supported version of the Reader application.) You can test PDF files that are displayed in a browser or in the stand-alone Adobe Reader application. A functional test script that is recorded for files that are displayed in the stand-alone Adobe Reader application can be played back when the file is displayed in a browser and vice-versa, provided that the script is recorded for document controls only. Support for Adobe PDF requires Visual C++ 2013 or higher.

Rational® Functional TesterHCL OneTest™ UI supports testing of PDF files by either interacting with specific document controls or through Reader controls.

The level of granularity that Rational® Functional TesterHCL OneTest™ UI supports depends on the way the PDF file is designed. For example, if the entire page of a PDF file is designed to contain one text object only, the verification point highlighter captures only the page-level contents and does not access the content inside the page.

Cross-compatibility of testing PDF read-only files is possible only for document controls and not Reader controls. For example:

- **Stand-alone Reader and browser:** A test script that is recorded on a PDF file that is opened in the stand-alone Reader can be played back on a PDF file opened in browser, provided that the script is recorded for document controls only.
- **Reader 7.0, 8.0, 9.0, 10.0 and 11.0:** A test script that is recorded on a PDF file that is opened in the stand-alone Adobe Reader 7.0 can be played back on a PDF file opened in Adobe Reader 8.0 and so on, provided that the script is recorded for document controls only. Additionally, a functional test script that is recorded for files that are displayed in the stand-alone PDF Reader can be played back when the document is displayed in a browser and vice-versa, provided that the script is recorded for document controls only, and provided that the regular expression is used for the top-level window.

Note: PDF 9.0, 10.0 and 11.0 file testing support:

- When you record on the Reader toolbar buttons in Adobe Reader 10.0 and 11.0, the action is recorded as click(atPoint(x,y)).
- When you record on a PDF file that is opened in an Internet Explorer browser, you must first record actions in the browser and then begin the recording on the PDF file.

Note: Only the Internet Explorer browser supports the recording of a PDF file.

- Playback fails when only the find() API is used to locate objects. As a workaround, click the captured object first and then play back.

Rational® Functional TesterHCL OneTest™ UI provides support for testing PDF files that contain the following document controls:

- Page
- Table
- Text
- Link
- Outline Tree
- Graphics
- Document

Rational® Functional TesterHCL OneTest™ UI provides support for testing PDF files that contain the following Reader controls:

- Button
- Check box
- Toggle button
- Text box
- Combo box

Prerequisites: Before you start testing PDF files, you must set the Adobe Reader preferences and the Rational® Functional TesterHCL OneTest™ UI script assure values.

- **Setting preferences for Adobe Reader 7.0**
  1. Open Adobe Reader 7.0.
  2. Click **Document > Accessibility Setup Assistant**.
  3. Click **Next** on the Accessibility Setup Assistant page.
  4. Select **Fit page** as the **Default display zoom** in **Screen 2 of 5**, and click **Next**.
  5. Clear **Confirm before adding tags to documents** in **Screen 3 of 5**, and click **Next**.
  6. In **Screen 4 of 5:**
     - For Page mode setting, select **Deliver currently visible pages**.
     - For Document mode setting, select **Deliver the entire document at once** .
  7. Click **Next**.
  8. In **Screen 5 of 5**, select **Display PDF documents in the web browser**.
  9. Click **Done**.

10. Click **Edit > Preferences** in Adobe Reader.
11. Click the **Page Display** category and select **Single Page** as the **Default Page Layout**, and click **OK**.

- **Setting preferences for Adobe Reader 8.0, 9.0, 10.0 and 11.0**

Note: For version 10.0, ensure that you disable protected mode before you set the preferences. For instructions, see the procedures **Disabling the protected mode for Adobe Reader 10.0** and **Disabling the protected mode for Adobe Reader 11.0**. For version 11.0, ensure that you disable Enhanced Security. Under **Edit > Preferences**, go to **Security(Enhanced)**, and uncheck **Enable Enhanced Security**.

1. Open Adobe Reader.
2. Click **Document > Accessibility Setup Assistant**.
3. Click **Next** twice.
4. Clear **Confirm before tagging documents** in **Screen 3 of 5** and click **Next**.
5. In **Screen 4 of 5**:
   - For Page mode setting, select **Only read the currently visible pages** in the **Page vs Document** field.
   - For Document mode setting, select **Read the entire document at once** in the **Page vs Document** field.
6. Click **Next**.
7. In **Screen 5 of 5**, select **Display PDF documents in the web browser**.
8. Click **Done**.

Note: Verify that **Click to show one page at a time** button at the top toolbar of the Adobe Reader is selected for recording and playing back.

- **Disabling protected mode for Adobe Reader 10.0**
  1. Click **Edit > Preferences**.
  2. Click **General** in the Categories list.
  3. Clear the **Enable Protected Mode at startup** check box.
  4. Click **OK**, and restart the reader.
  5. Set the accessibility options as described in the **Setting preferences for Adobe Reader 8.0, 9.0, 10.0 and 11.0** procedure.
- **Disabling protected mode for Adobe Reader 11.0**
  1. Click **Edit > Preferences**.
  2. Click **Security (Enhanced)** in the Categories list.
  3. Clear the **Enable Protected Mode at startup** check box in the Sandbox Protection Area. When prompted to confirm your choice, click **OK**.
  4. Click **OK**, and restart the reader.
  5. Set the accessibility options as described in the **Setting preferences for Adobe Reader 8.0, 9.0, 10.0 and 11.0** procedure.
- **Setting the Rational® Functional TesterHCL OneTest™ UI script assure values**

1. Open Rational® Functional TesterHCL OneTest™ UI, and click **Window > Preferences**.
2. In the **Preferences** window, expand **Functional Test > Playback**, and then click **Script Assure**.
3. Click **Advanced**.
4. Specify the following values on the Script Assure page:
    - **Maximum acceptable recognition score**: `5000`
    - **Last chance recognition score**: `10000`
    - **Ambiguous recognition scores difference threshold**: `1000`
    - **Warn if accepted score is greater than**: `5000`

 **Notes:**

- When you open a new file in Adobe Reader, click the PDF file a few times after the recorder starts, till a meaningful recording statement is seen in the recording monitor. These clicks are required for the Reader to process the file.
- Only document verification points are supported in the document mode setting. A verification point can be taken after the mandatory click in the file. Any clicks on the PDF document in document mode setting while recording are ignored and no code is generated.
- Use the Page mode setting for larger documents.
- To take data verification point on a large document in Document mode, follow these steps:
    1. Create the following registry key:
        - HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rational Test\8\Options
        - HKEY_LOCAL_MACHINE\SOFTWARE\HCL Technologies\HCL OneTest UI\Options
    2. Create a new DWORD variable `InvocationTimeout`.
    3. For documents of 70 pages or more, specify the timeout value of `8*60*1000` milliseconds.
    4. Restart Rational® Functional TesterHCL OneTest™ UI and restart Adobe Reader.
    5. Perform the clicks on the PDF file until this message is generated: "Ignoring the click on PDF document in document mode setting".
    6. Take the data verification point.
- If a particular control in the PDF file spans over two or more lines, the highlight rectangle covers all the lines in that control location. Other controls might fall in the highlight rectangle. But when a verification point is taken on the control that spans over two or more lines, any other controls that fall within the screen rectangle are not considered.
- You might not be able to test the PDF files correctly if the font of the letters in the PDF file is not available or installed on the computer.

Related reference

Task flow for testing applications on page 47

AJAX support

Rational® Functional TesterHCL OneTest™ UI supports testing AJAX-based web applications.

You can test AJAX-based applications in two different ways; by setting the Auto Trace option to true or by setting the Auto Trace option to false.

APIs for testing AJAX-based applications

The following APIs can be used in functional test scripts for testing AJAX-based applications. These APIs must be invoked on the HTML.Document test objects.

Method	Description	Example
SetAjaxTrace(boolean)	To trace the AJAX requests on the Document control	`document_htmlDocument().setAjaxTrace(true);`
GetAjaxPendingRequests ()	Returns the number of AJAX pending requests at any given point of time since the first AjaxTraceOn.	`document_htmlDocument().getAjaxPendingRequests();`
WaitForAjaxPendingRequests (int)	To wait for the specified number of AJAX requests to be completed. Waits indefinitely till pending requests becomes zero, if the argument is not specified.	`document_htmlDocument().waitForAjaxPendingRequests(2);`
GetAjaxCompletedRequests ()	Returns the number of AJAX requests completed at any given point of time since the first AjaxTraceOn..	`document_htmlDocument().getAjaxCompletedRequests();`
WaitForAjaxCompletedRequests (int)	To wait for the specified number of Ajax requests to be completed.	`document_htmlDocument().waitForAjaxCompletedRequests(4);`

Note: For more information on these APIs, see the API Reference topics.

Related reference

Task flow for testing HTML applications on page 51

Application Response Measurement support

The Application Response Measurement (ARM) standard helps measure the end-to-end transaction performance of an application, including service levels and response time. Rational® Functional TesterHCL OneTest™ UI uses the response time breakdown to view the statistics that is captured while running a test.

Response time breakdown

Response time breakdown is a type of application monitoring that shows how much time was spent in each part of the system under test. The response time breakdown view shows the "insides" of the system under test.

To capture response time breakdown data, you must enable it in Rational® Functional TesterHCL OneTest™ UI. The data collection infrastructure collects response time breakdown data. Each host on which the application runs and from which you want to collect data must have the data collection infrastructure installed and running.

Note: Ensure that you use a compatible version of the response time breakdown tool such as IBM® Tivoli® Composite Application Manager (ITCAM) for Rational® Functional TesterHCL OneTest™ UI to integrate with ARM. For information about using the response time break down tool and the versions, refer the respective product information centre.

Related reference

Task flow for testing applications on page 47

Dojo support

IBM® Rational® Functional TesterHCL OneTest™ UI supports testing HTML applications that contain Dojo controls on versions of all browsers supported by Rational® Functional TesterHCL OneTest™ UI. You can test applications that are developed using Dojo Toolkit versions 1.0, 1.1, 1.2, 1.3.2, 1.4.2, 1.5, 1.6.1, 1.7, 1.8, and 1.9.

Rational® Functional TesterHCL OneTest™ UI supports recording and testing HTML applications that are developed using the Dojo Toolkit versions 1.0, 1.1, 1.2, 1.3.2, 1.4.2, 1.5, 1.6.1, 1.7, 1.8, and 1.9 containing Dojo controls as shown in Table 1:

Note: On Mozilla Firefox browsers, Dojo recordings on the Dojo Grid Control are incorrect for rows beyond 25. This happens because Dojo restarts its row index from zero every time a row beyond row 25 is clicked. Use one of the following workarounds to overcome this problem:

- Modify the recorded script and mention the correct row number.
- Record the script in an Internet Explorer browser, and play it back on a Mozilla Firefox browser.

Table 58. Dojo controls supported by FT against Dojo Toolkit versions

Versions 1.0, 1.1, 1.2, 1.3.2 and 1.4.2	Version 1.5	Versions 1.6.1, 1.7, and 1.8	Version 1.9
• Button • Calendar • Checkbox	• Button • Calendar • Checkbox	• Button • Calendar • Checkbox	• Button • Calendar • Checkbox

Table 58. Dojo controls supported by FT against Dojo Toolkit versions (continued)

Versions 1.0, 1.1, 1.2, 1.3.2 and 1.4.2	Version 1.5	Versions 1.6.1, 1.7, and 1.8	Version 1.9
• ColorPalette • Combobutton • Currencytextbox • Datetextbox • Dialog • Dropdown Combobox • Dropdownbutton • Editor • Grid • InlineEditBox • Menu • Numbertextbox • Slider • Spinner • Textarea • Textbox • Timetextbox • Toolbar • Tooltip • TooltipDialog • Tree • Validationtextbox • Dojo containers: ◦ Accordian ◦ Content Pane ◦ Floating Pane ◦ Layout ◦ Link Pane ◦ Split ◦ Stack ◦ Tab ◦ Title Pane	• ColorPalette • Combobutton • Currencytextbox • Datetextbox • Dropdown Combobox • Editor • Grid • Menu • Numbertextbox • Textarea • Textbox • Timetextbox • Toolbar • Tree • Dojo containers: ◦ Accordian ◦ Content Pane ◦ Floating Pane ◦ Layout ◦ Link Pane ◦ Split ◦ Stack ◦ Tab ◦ Title Pane	• ColorPalette • Combobutton • Currencytextbox • Datetextbox • Dropdown Button • Dropdown Combobox • Editor • Grid • InlineEditBox • Menu • Numbertextbox • Slider • Spinner • Textarea • Textbox • Timetextbox • Toolbar • TooltipDialog • Tree • Validationtextbox • Dojo containers: ◦ Accordian ◦ Content Pane ◦ Floating Pane ◦ Layout ◦ Link Pane ◦ Split ◦ Stack ◦ Tab ◦ Title Pane	• ColorPalette • Combobutton • Currencytextbox • Datetextbox • Dropdown Button • Dropdown Combobox • Editor • Grid • InlineEditBox • Menu • Numbertextbox • RadioMenuItem • Slider • Spinner • Textarea • Textbox • Timetextbox • Toolbar • TooltipDialog • Tree • Validationtextbox • Dojo containers: ◦ Accordian ◦ Content Pane ◦ Fieldset ◦ Floating Pane ◦ Layout ◦ Link Pane ◦ Split ◦ Stack ◦ Tab ◦ Title Pane

Related reference

Task flow for testing HTML applications on page 51

Eclipse support

Rational® Functional TesterHCL OneTest™ UI supports testing of applications based on Eclipse 3.0.x, 3.1.x, 3.2.x, 3.3.x, 3.4.x, 3.5.x, 3.6.2, 4.2, 4.2.2, and 4.4.

You must enable the Eclipse platform for testing before recording scripts to test Eclipse-based applications.

Note: For specific information on the versions that are tested, refer the technical document available at Software Product Compatibility Reports

You can test both Eclipse 32-bit and 64-bit Standard Widget Toolkit (SWT) and Rich Client Platform (RCP) applications using Rational® Functional TesterHCL OneTest™ UI.

Rational® Functional TesterHCL OneTest™ UI also supports testing of stand-alone SWT applications. To do this, you must enable the JRE in which the SWT application runs, and also enable the SWT application. For information about enabling SWT applications, see Enabling stand-alone Standard Widget Toolkit applications on page 795.

Note: For Eclipse versions 4.2 and 4.2.2, only 32-bit applications are supported.

Related reference

Task flow for testing Eclipse applications on page 53

Flex support

IBM® Rational® Functional TesterHCL OneTest™ UI supports testing functional aspects of Adobe® Flex applications. You can record and playback scripts against Flex-based user interfaces inside a web browser and verify that the application functions correctly.

You must consider the following points before testing with Adobe Flex applications:

- Testing Flex applications are supported on Microsoft Internet Explorer browsers only.
- While taking a data verification point on multiple selected elements of a Flex list control, not all selected values are picked up.
- Insert Test Object does not work for Flex and SparkFormControl.
- It is not possible to play back actions that are recorded on the vertical scroll bar of a SparkList.

You can test the following list of controls in the Flex applications:

- FlexAccordion
- AdvancedDataGrid
- Alert
- Application
- Box
- ButtonBar
- Button
- ColorPicker
- Combobox
- CheckBox
- Container
- DataGrid
- DateChooser
- DividedBox
- Label
- LinkBar
- List
- MenuBar
- Menu
- NumericStepper
- OLAP Datagrid
- Panel
- PopUpbutton
- Progressbar
- RadioButton
- RichTextEditor
- ScrollBar
- Slider
- TabNavigator
- TextArea
- TitleWindow
- ToggleButtonBar
- Tree

Testing Flex 4.0, 4.1 and 4.5 applications that contain the following Spark controls is supported:

- Application
- BorderContainer
- Button
- ButtonBar
- ButtonBarButton
- CheckBox

- ComboBox
- DataGroup
- DropDownList
- Label
- List
- ListLabel
- MuteButton
- NumericStepper
- Object
- Panel
- RadioButton
- RichEditableText
- RichText
- ScrollBar
- Scroller
- SkinnableContainer
- SkinnableContainerBase
- Slider
- Spinner
- TabBar
- TextArea
- TextBase
- TextInput
- TileGroup
- TitleWindow
- ToggleButton
- VideoPlayer
- VolumeBar

In addition to the controls listed above, the following controls are supported for Flex 4.5:

- SparkDataGrid
- SparkForm
- OSMFVideoPlayer

Playing back Flex scripts in Internet Explorer 11

You must recompile Flex applications (AUT) using the command-line compiler (.swc files) and the appropriate Flex Builder SDK, if you want to use Internet Explorer 11 in the following scenarios:

- Playing back the already existing Flex application test scripts that you recorded on an older version of Internet Explorer.
- Recording new test scripts and playing them back.

For details, see Configuring Flex application using tools on page 815.

Related reference

Task flow for testing Flex applications on page 57

Flex custom control support

IBM® Rational® Functional TesterHCL OneTest™ UI supports testing functional aspects of Adobe® Flex custom controls in a generic and specific way.

About this task

Rational® Functional TesterHCL OneTest™ UI support Flex custom controls in two different ways:

- **Generic support:** Rational® Functional TesterHCL OneTest™ UI supports recording and playback of scripts. Recording is generic and methods `performAction(eventname, "arg1""arg4");`. For example, `flex__randomWalk_RandomWalk1().performAction("Select", "Food");`

  Rational® Functional TesterHCL OneTest™ UI cannot capture data verification point and does not support data-driven testing in generic support. Data verification point is achieved using the `getProperty()` method and by verifying the value. For example:

  ```
  String selectedItem = (String)list__randomWalk_RandomWalk1().getProperty("selectedItem");
  String verificationData = "TestVerify";
  if(selectedItem.equalsIgnoreCase(verificationData))
  {
  //code to do
  }
  ```

  Data-driven testing is supported by using an action and associating the argument value with a dataset. Since data-driving is control specific, a generic method is not available using drag-hand. For example,

  ```
  flex__randomWalk_RandomWalk1().performAction("Select",dpString( variableName/index));
  ```

  For Rational® Functional TesterHCL OneTest™ UI to support Flex custom control in a generic way:

  1. You must write a delegate for the custom control. Delegate is an actionscript class which allows automation framework to understand the events from the control. References are available in Flex Builder directory where a delegate exists corresponding to each standard control. For more information see, Flex Data Visualization Developer's Guide in the Adobe site.
  2. Map the custom control with its events and properties in the `FlexEnv.xml` file located in the `bin` folder of the Rational® Functional TesterHCL OneTest™ UI installation directory.
  3. Every new custom control is mapped to the base proxy `flexObjectProxy`, and base test objects are mapped to `FlexObjectTestObject`.
- **Specific support:** Proxy and test objects are created. The proxy is mapped to the control in the .rftcust file that is generated while creating the proxy using the proxy SDK wizard. Data verification point and data drive is created in the new proxy as required. Role are assigned to the control. Recognition properties are added to the control.

GEF support

Rational® Functional TesterHCL OneTest™ UI allows testing the functionality of GEF objects that are implemented using standard GEF editors and non-standard GEF editors. GEF editor is based on Eclipse Modelling Framework (EMF) that provides many features for manipulating models. GEF displays a model graphically, employs MVC (model-view-controller) architecture and aids user interaction with that model.

You can perform data, image or properties verification point on the GEF objects since Rational® Functional TesterHCL OneTest™ UI recognizes the GEF EditParts and Palettes. You must enable the GEF applications for testing before recording the functional test scripts. If the GEF application is not enabled for testing, Rational® Functional TesterHCL OneTest™ UI recognizes GEF objects as FigureCanvas.

Note: If the object recognition is weak for any of the GEF objects, you can modify the recognition properties for these objects (org.eclipse.gef.editparts.AbstractEditPart, org.eclipse.gef.palette.PaletteEntry) in the object library.

Related reference

Task flow for testing Eclipse applications on page 53

HTML support

IBM® Rational® Functional TesterHCL OneTest™ UI supports testing HTML applications that are loaded in a browser.

The Functional Test perspective can be used to test HTML4 applications and the UI Test perspective can be used to test web applications built with HTML4 and HTML5 technologies. For information on testing of HTML5-based applications, refer to Testing web applications on page 532.

Rational® Functional TesterHCL OneTest™ UI supports the following options to test HTML applications:

- Recording tests for HTML applications using the different browser settings. See Web UI recording on page 532.
- Using a specific browser profile for test runs. See Setting a specific browser profile for the playback on page 1236.
- Clearing the data stored in browsers before test runs. See Clearing cache, cookies, and history of browsers on page 1237.
- Running tests in headless and private mode. See Running tests in the headless mode on page 1257 and Running tests in the private or incognito mode on page 1259.
- Updating tests with guided healing and self-healing. See Overview of guided healing and self-healing on page 617.
- Multi-window support to test Functional HTML tests. See Multiwindow support for functional HTML tests on page 784.

Rational® Functional TesterHCL OneTest™ UI supports testing HTML applications that are loaded in the following browsers:

- **Google Chrome**: See Preparing for functional testing in the Google Chrome browser on page 784.
- **Microsoft® Edge**: See Running a script from the Microsoft Edge browser on page 1449.
- **Internet Explorer**
- **Mozilla Firefox**: See Enabling Mozilla Firefox for Web UI testing and Recording a Web UI test in Mozilla Firefox on page 553.

When you record a script, Rational® Functional TesterHCL OneTest™ UI creates a test object map for the application under test. The test object map contains descriptions of all test objects to which the script refers. The test object maps that Rational® Functional TesterHCL OneTest™ UI creates for HTML applications are often more hierarchical than those created for Java™ applications. The highest level of the test object map is a browser and the HTML application is inside the browser. For more information, see the example of a test object map that is created for a HTML application.

Related reference

Task flow for testing HTML applications on page 51

Automatically enabled environment for functional testing on page 774

Related information

Tips and tricks for functional testing HTML applications on page 1543

Playback of test scripts for HTML applications by using the Web UI engine on page 1447

Enabling web browsers on page 780

Java support

This functional testing application supports the following Java™ versions for testing Java™ applications and HTML applications with applets.

Table 59. Java support

32-bit mode	64-bit mode
• IBM JRE 1.6 • IBM JRE 1.7 • IBM JRE 1.8 • Sun JRE 1.6 • Oracle JRE 1.7 • Oracle JRE 1.8 • OpenJDK 8	• IBM JRE 1.7 • IBM JRE 1.8 • Sun JRE 1.6 • Oracle JRE 1.7 • Oracle JRE 1.8 • Oracle JRE 9 • Oracle JRE 10 • Oracle JDK 11 • OpenJDK 8

Table 59. Java support (continued)

32-bit mode	64-bit mode
	Note: HTML applications with applets can be tested in Java 8 and earlier but this is not supported in Java 9 and later.

For specific information on the recommended versions that are validated and have provided optimal results, refer the technical document available at Software Product Compatibility Reports

---

Related reference

Task flow for testing Java applications on page 49

.NET support

IBM® Rational® Functional TesterHCL OneTest™ UI supports recording and playing back user applications that are built with the Microsoft .NET Framework. You can also test Microsoft Windows 32-bit applications that contain .NET controls.

Testing .NET 3.5 applications: Supported controls

Rational® Functional TesterHCL OneTest™ UI supports testing .NET 3.5 applications that contain these Windows Presentation Foundation (WPF) controls:

- Layout:
  - BulletDecorator
  - Canvas
  - DockPanel
  - Expander
  - Grid
  - GridSplitter
  - GroupBox
  - Panel
  - ScrollBar
  - ScrollViewer
  - StackPanel
  - Thumb
  - ViewBox
  - Window
- Buttons
  - Button
  - RepeatButton

- Menus
    - ContextMenu
    - Menu
    - ToolBar
- Selection
    - CheckBox
    - ComboBox
    - ListBox
    - ListView
    - TreeView
    - RadioButton
    - Slider
- Navigation
    - NavigationWindow
    - TabControl
- User Information
    - Label
    - PopUp
    - ProgressBar
    - StatusBar
    - TextBlock
- Documents
    - DocumentViewer
    - FlowDocumentPageViewer
    - FlowDocumentScrollViewer
- Input
    - TextBox
    - RichTextBox
    - PasswordBox
- Media
    - Image

- Digital Ink
    - InkCanvas

Testing .NET 4.0 applications: Supported controls

In addition to the supported controls for .NET 3.0 and 3.5 applications, Rational® Functional TesterHCL OneTest™ UI supports testing .NET 4.0 applications that contain these Windows Presentation Foundation (WPF) controls:

- Data display
    - DataGrid
- Date display and selection

- Calendar
- DatePicker

Testing .NET 4.5 applications: Supported controls

In addition to the supported controls for .NET 3.0, 3.5 and 4.0 applications, Rational® Functional TesterHCL OneTest™ UI supports testing .NET 4.5 applications that contain these Windows Presentation Foundation (WPF) controls:

Note: Recording actions on the drop-down list of a RibbonFilterMenu is not supported.

- Ribbon controls provided by the System.Windows.Controls.Ribbon namespace:
  - System.Windows.Controls.Ribbon.RibbonApplicationMenu
  - System.Windows.Controls.Ribbon.RibbonApplicationMenuItem
  - System.Windows.Controls.RibbonApplicationSplitMenuItem
  - System.Windows.Controls.RibbonButton
  - System.Windows.Controls.RibbonCheckBox
  - System.Windows.Controls.RibbonComboBox
  - System.Windows.Controls.RibbonContextMenu
  - System.Windows.Controls.RibbonMenuButton
  - System.Windows.Controls.RibbonMenuItem
  - System.Windows.Controls.RibbonQuickAccessToolBar
  - System.Windows.Controls.RibbonRadioButton
  - System.Windows.Controls.RibbonSplitButton
  - System.Windows.Controls.RibbonSplitMenuItem
  - System.Windows.Controls.RibbonTab
  - System.Windows.Controls.RibbonTabHeader
  - System.Windows.Controls.RibbonTextBox
  - System.Windows.Controls.RibbonToggleButton
  - System.Windows.Controls.RibbonToolTip
  - System.Windows.Controls.RibbonWindow

Rational® Functional TesterHCL OneTest™ UI supports testing .NET 4.0 and 4.5 applications hosting side by side CLRs.

Testing .NET 4.5.1 applications

The Microsoft .NET 4.5.1 Framework does not offer any new UI controls. However, Rational® Functional TesterHCL OneTest™ UI continues to support all the controls (listed as supported so far) contained in the .NET 4.5.1 applications.

Related reference

Task flow for testing applications on page 47

Windows support

Rational® Functional TesterHCL OneTest™ UI supports testing of Windows applications with Win32 controls.

Rational® Functional TesterHCL OneTest™ UI supports testing of the following list of Win32 controls:

- Button
- Calendar
- CheckBox
- ComboBox
- ComboListBox
- DateTimePicker
- Edit ListBox
- ListView
- Menubar
- PopUpMenu
- Radio Button
- Rebar
- RichEdit
- ScrollBar
- sysMonthCalendar
- TabControl
- ToggleButton
- ToolBar
- ToolTip
- TrackBar
- TreeView
- StatusBar

Rational® Functional TesterHCL OneTest™ UI supports testing of the following list of Microsoft® Foundation Class (MFC) controls:

- Animate
- Button
- Menu
- DateTimePicker
- Calender
- Combobox
- Listbox
- Edit
- RichEditBox
- Progressbar
- Tab

- Trackbar (slider)
- Tree
- Header
- Hotkey
- SpinButton
- ListView
- Checkbox
- StaticText
- Scrollbar

Related reference

Task flow for testing applications on page 47

Nested domains support

Rational® Functional TesterHCL OneTest™ UI recognizes two different kinds of nesting of objects, a parent-child nesting and an owner-owned nesting. A parent-child nesting occurs when one object is contained within another, such as a button on a form. An owner-owned nesting occurs when the owned object has its own top-level window, such as a dialog box that is owned by a top-level window.

Rational® Functional TesterHCL OneTest™ UI supports some instances of nesting of objects from different domains. That means you can test an object of one domain that is nested inside an object of another domain, and Rational® Functional TesterHCL OneTest™ UI will accurately understand the objects and their domains. If a nesting of one domain within another is not supported, Rational® Functional TesterHCL OneTest™ UI will model the objects consistently but may not accurately understand the domain for the nested objects. For example, Rational® Functional TesterHCL OneTest™ UI does not support the nesting of a .Net control within a Windows® application. In this case, Rational® Functional TesterHCL OneTest™ UI is likely to see the .Net controls as if they were Windows® controls (because .Net controls are often implemented using the underlying mechanisms of Window controls). Another example is a Windows-based dialog that appears on top of a Java™ application. In this case, the Java™ domain does not understand the windows dialogs, or even acknowledge that they exist. And since by default Rational® Functional TesterHCL OneTest™ UI does not dynamically enable a Java™ application as if it were a Windows® application, there is no Windows® domain in that process. So in this instance, the dialogs are not testable objects without scripting.

The following list describes the cases of nested domains that are supported.

HTML - ActiveX as child -- This is an HTML page that contains ActiveX controls.

HTML - Windows® Owned -- Some common dialogs displayed by Internet Explorer will appear as Windows® domain objects.

.Net - ActiveX as child -- This is a .Net Winforms application that utilizes legacy ActiveX controls.

.Net - HTML as child -- This is a .Net WinForms application that utilizes an embedded Internet Explorer browser control.

.Net - HTML as owned -- This is .Net WinForms application that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

.Net - Windows® as owned -- This a .Net WinForms application that displays some form of non-WinForm dialog. For example, when a .Net application displays a common dialog (File Open, Print, etc.) or messagebox.

Java™ - HTML as child -- This a Java™ SWT application that utilizes an embedded Internet Explorer browser control.

Java™ - HTML as owned -- This a Java™ SWT application that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

Windows® - HTML as child -- This is a generic Windows® application (possibly VB 6.0 or MFC) that utilizes an embedded Internet Explorer browser control.

Windows® - HTML as owned -- This is a generic windows application (possibly VB 6.0 or MFC) that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

Windows -.Net as child – This is a generic Windows® application (possibly VB 6.0 or MFC) that embeds .Net WinForm controls.

Nested domains in the object map

You can see that objects are of different test domains by looking at the object map. In the object map, each object is listed by its object type and domain type. For example, the following object:

Java™: Button: close-order: javax.swing.JButton

is a Java™ button that is a javax.swing.JButton object type. The "Java™:" prefix shows that the object is in the Java™ test domain. The test domain of every object is always the first thing shown on each object listed in the map, as shown in the example above. If a child object has a different domain than the parent object, you will see two different prefixes in their entries in the object map.

Related reference

Task flow for testing applications on page 47

PowerBuilder support

IBM® Rational® Functional TesterHCL OneTest™ UI supports the testing of Win32 targets and .NET targets that are created by using Sybase PowerBuilder versions 10.5, 11.0, 11.2, 11.5, 12.0, 12.5, 12.6, and Appeon PowerBuilder 2017.

Rational® Functional TesterHCL OneTest™ UI supports testing these controls:

- CheckBox
- CommandButton
- Contemporary menu (For a Win32 target, supports PowerBuilder 11.2 with EBF 8786 and PowerBuilder 11.5.1 with EBF 4526)
- DatePicker
- DataWindow (Presentation styles: Grid, Tabular, and Freeform. For supported freeform controls, see the list further down this page)
- DropdownListBox
- DropDownPictureListBox
- EditMask
- Groupbox
- HProgressBar
- HScrollBar
- HTrackBar
- ListBox
- ListView
- Menu
- MonthCalendar
- MultiLineEditor
- Picure
- PictureButton
- PictureHyperLink
- PictureListBox
- RadioButton
- SingleLineEdit
- StaticHyperLink
- StaticText
- Tab
- TreeView
- VProgressBar
- VScrollBar
- VTrackBar
- Window

These controls are supported for a freeform DataWindow for Win32 targets:

- Bitmap (for pictures)
- Button
- Column



Note: In a freeform DataWindow, columns are identified with their edit styles. Only these edit styles are supported:



- CheckBox
- DropDownListBox (individual item selection is not supported but the coordinates of the group are recorded)
- DropDownDataWindow
- RadioButton (individual item selection is not supported but the coordinates of the group are recorded)
- Text

- Compute (for computed field)
- DataWindow
- Graph
- GroupBox
- Label
- OLE objects
- Shapes:
  - Ellipse
  - Rectangle
  - Roundrectangle

Data verification points are supported only for these controls for a freeform DataWindow for Win32 targets:

- Button
- Column (only for these edit styles):
  - DropDownDataWindow
  - DropDownListBox
  - Text
- GroupBox
- Label

Related reference

Task flow for testing applications on page 47

SAP support

IBM® Rational® Functional TesterHCL OneTest™ UI supports extended functional testing of SAP GUI for Windows and SAP GUI for HTML.

Rational® Functional TesterHCL OneTest™ UI supports testing of all SAP R/3 versions through the SAPGUI running on Windows®. For information on the supported versions of SAP GUI, refer to Recording and playback support on page 22.

With this support, you can use Rational® Functional TesterHCL OneTest™ UI to record and play back scripts against the SAP UI with reliable recognition against SAP controls, including customized data verification of SAP controls. Rational® Functional TesterHCL OneTest™ UI support is built on top of SAP GUI scripting framework exposing all scripting capabilities provided by SAP as well as adding significant value through the inherent capabilities of Rational® Functional TesterHCL OneTest™ UI.

Enabling SAP support for pre-existing projects in the Visual Studio IDE

To use the SAP support for a project that was created using the releases of IBM® Rational® Functional TesterHCL OneTest™ UI version 6, you must add two new templates (one for script headers and one for script helper headers) to the project. You will also need to update the build or reference information for the project.

1. In the Solution Explorer, right-click the project and select the **Properties** option.
2. In the Properties dialog box, select the **Functional Test Script Templates** tab.
3. Select the template type **Script Helper: Header of the file**.
4. If you have not customized this template, you can upgrade it by clicking the **Restore Defaults** button.
5. Add the line `Imports Rational.Test.Ft.Object.Interfaces.SAP` in the import section of the template.
6. After modifying the template, click the **Apply** button.
7. Select the template type **Script: Header of the file** and add the same line in the import section of the template.
8. Select the two templates**ft_script.vb.rfttpl** and **ft_scripthelper.vb.rfttpl** , right-click and select the **Check in** option.

   Your other team members can update their project by right-clicking the project in the Solution Explorer and clicking **Team > Get Latest Version**.
9. Finally, right-click the project again in the Solution Explorer and click **Reset Custom Reference**.

   Each member of your team must perform this last step, as the Custom References is local to each project on each machine.

Visual Basic support

Rational® Functional TesterHCL OneTest™ UI supports testing Visual Basic 5.0 and 6.0 applications.

You can test the following controls in Visual Basic applications:

- Form
- MDIForm
- CheckBox
- ComboBox
- CommandButton
- Data
- DirectoryListBox
- DriveListBox
- FileListBox

- Frame
- Image
- Label
- Line
- ListBox
- OLEContainer
- OptionButton
- PictureBox
- Shape
- TextBox

Related reference

Task flow for testing applications on page 47

Terminal-based applications support

Rational® Functional TesterHCL OneTest™ UI Extension for Terminal-based Applications supports functional testing of Mainframe or zSeries (TN3270, TN3270E), AS/400 or iSeries (TN5250) and pSeries or Virtual Terminals (VT default, VT100, VT101, VT102, VT220-7, VT220-8, VT320, VT420-7, VT420-8, VT52, VT UTF-8).

Rational® Functional TesterHCL OneTest™ UI Extension for Terminal-based Applications tool helps you create test scripts to automate the host application test cases.

It provides a rich set of capabilities to test host attributes, host field attributes and screen flow through a host application. It uses terminal verification points and properties, as well as synchronization code to identify the readiness of terminal for user input.

You can use this tool to perform the following tasks:

- Store, load, and share common host configurations by using a properties file. The connection configuration can be loaded automatically through scripts, using these files
- Record or play back scripts against multiple host terminals.
- Start the terminal even when you are not recording or playing back your scripts. With this function, you can interact with the host without leaving the working Eclipse environment.
- Perform data driven testing.

Command line interface

In this section, you will learn about the actions that can also be performed from the command line.

These actions include:

- Record a script
- Compile a script

- Play back a script and pass command-line arguments to the script
- View and edit verification point and object map files
- Invoke the Java/HTML enabler
- Invoke the Application Configuration Tool

Core command line format

The core command line format is as follows:

java *<standard java options>* `-classpath` `rational_ft.jar` `hcl_ft.jar` `com.rational.test.ft.rational_ft` *<product options>* or

java *<standard java options>* `-jar` `rational_ft.jar hcl_ft.jar` *<product options>*

The *standard java options* refer to the Java™ command line options such as **-classpath <classpath/>** to set the classpath appropriately.

-classpath

If you use the first command line format, you must explicitly include the `rational_ft.jar` `hcl_ft.jar` in the classpath. It can be found in the `<productInstallationDirectory>` directory. If you use the second command-line format, specify the full path of the `rational_ft.jar` `hcl_ft.jar` file after the `-jar` option. You do not need to specify a classpath or the class to run (com.rational.test.ft.rational_ft). If you are using the product with PurifyPlus™, use the first command line format.

Note: If the external jar file or directory that is referenced in a project, does not meet the project path criteria, then such jar files must be added to the classpath.

See the API Reference for a full list of the command line options.

-projectpath

If you have dependencies on a project, then you must add the project path using the `-projectpath` option. If you have added other project dependencies to the functional test project, you can specify the project name using the `-projectpath` option. For example, Consider a scenario where you have a functional test script that uses a class from another project. To resolve this dependency, during a command line invocation, you must specify the dependent project name using the `-projectpath` option.

In a scenario where you have added external jar files or projects to a Rational® Functional TesterHCL OneTest™ UI project and the playback for this is run from the command line prompt then add the jar file to the project path provided the jar file contains a Rational® Functional TesterHCL OneTest™ UI project or the project that is added is a Rational® Functional TesterHCL OneTest™ UI project.

Commonly used command line options

The following table lists the more commonly used command line options:

Parameter	Description
-appconfig	Use this option to open the **Application Configuration Tool** dialog box.
-compile	Use this option to compile the test script for the first time before playback.
-compileall	Use this option to compile all test scripts in the project for the first time before playback. If you use the *-datastore* option, you can compile all tests scripts in the datastore directory and subdirectories. (Requires 9.1.1.1 and newer version of Rational® Functional TesterHCL OneTest™ UI)
-datastore <datastore directory>	Use this option whenever a script is specified. For example, use it with `-record` or `-playback`
-edit <file> or *-display <file>*	Use this option to edit or view a verification point or object map. The *<file>* can be a complete file name (with directory path). Use double-quotes if the name or path includes space characters.
-enable	Use this option to open the **Enable Environments** dialog box to enable a specific environment.
-exportlog	Optional. You can use this parameter to specify the file directory path to store the exported test log. For example, `java -jar "C:\Program Files\IBM\SDP\FunctionalTester\bin\rational_ft.jar" -datastore "C:\Users\user\IBM\rationalsdp\workspace\FunctionalTesterDemo" -playback "ClassicsJava_FindBasedScript" -exportlog "C:\result\temp_rft01" java -jar "C:\Program Files\HCL\HCLOneTest\FunctionalTester\bin\hcl_ft.jar" -datastore "C:\Users\user\HCL\hclonetest\workspace\FunctionalTesterDemo" -playback "ClassicsJava_FindBasedScript" -exportlog "C:\result\temp_rft01"`
-exportReport	Use this option to export the unified report of Functional tests to the file formats such as PDF, HTML, and XML. **Note:** The exported XML file is a JUnit XML file. You can view this file in applications that support JUnit reporting formats. The command syntax is as follows: exportReport "type=<reporttype>;format=<file type1,file type2,file type3>;folder<destination folder path>;filename=<name of the exported file>

Parameter	Description
	For example, to export the report to only the pdf format, you can use exportReport "type=unified;format=pdf;folder=Exportedreport102;filename=testreport If you want to export the report to multiple formats, you can specify the file formats as comma-separated values. The file type value can be in uppercase or lowercase. For example, to export the report to all the supported formats, you can use exportReport "type=unified;format=pdf,xml,html;folder=Exportedreport102;filename=testreport The report in different file formats use the same file name that is specified in the command.
-inspector	Use this option to open the **Test Object Inspector Tool** dialog box.
-playback <script name>	Use this option to play back a Java™ script. You must specify the playback option at the end of the command. Rational® Functional TesterHCL OneTest™ UI ignores any arguments specified after the playback option.
-record <script name>	Use this option to record a new script (or in conjunction with `-insertafter <line number>` to insert recording into an existing script). .
-rt.log_format	Optional. Use this option to specify the format of the report that you want to generate after the playing back of a test. You can specify any of the following log types as the format of the report: • none • text • html • TPTP • xml • Default • Json **Restriction:** If you use *Default* as the log type to generate unified report for Functional test scripts from the command line, the report does not open automatically when the play back is complet-

Parameter	Description
	ed. You must use a different log type if you want the report to open automatically when the play back is completed. **Tip:** You can use the *-exportlog* parameter to export the test log in the Default log type format.

Note: The script name in the above options is not a file name. It is a fully qualified class name using the dot (.) character to separate package/namespace and script class name. You can use -<option> <script name> to record a Java™ or VB.NET script, depending on the project type.

For example, you can use -record test.script1 to create the `test` folder in a project and create the `script1` Java™ script in the folder.

Command line usage in Linux®

The command line format is `<Install_Directory>/jdk/jre/bin/java <Install_Directory>/FunctionalTester/bin/rational_ft.jar <product options><Install_Directory>/jdk/jre/bin/java <Install_Directory>/FunctionalTester/bin/hcl_ft.jar <product options>`. If the product is installed in the default path, then the you can use `/opt/IBM/SDP/jdk/jre/bin/java /opt/IBM/SDP/FunctionalTester/bin/rational_ft.jar <product options>/opt/HCL/HCLOneTest/jdk/jre/bin/java /opt/HCL/HCLOneTest/FunctionalTester/bin/hcl_ft.jar <product options>`.

If you use the above command-line format, you must explicitly set the functional test environment variables. Alternatively, you can use the script `ft_cmdline`.

Exemple

Examples:

Enable all environments in Linux®

```
/opt/IBM/SDP/ft_cmdline -enable ALL
```

```
/opt/HCL/HCLOneTest/ft_cmdline -enable ALL
```

Execute a script with command line arguments on Linux®

```
/opt/IBM/SDP/ft_cmdline -datastore /home/user/IBM/rationalsdp/workspace/Project1 -log testscript
 -playback Script2 -args arg1 arg2
```

```
/opt/HCL/HCLOneTest/ft_cmdline -datastore /home/user/HCL/hclonetest/workspace/Project1 -log testscript
 -playback Script2 -args arg1 arg2
```

Command line usage examples

A list of examples for some of the tasks that can be performed using the command line interface.

In these examples, `-classpath` must point to the rational_ft.jar hcl_ft.jarfiles.

Note: The <script-name> values use standard Java™ package or .NET namespace naming conventions such as `package.MyScript` or `Namespace.MyScript`.

Table 60.

Task	Example
Record a new script:	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [**-map** <sharedmap>] [options] **-record** <script-name>
Record a simplified script	java **-classpath** <classpath> com.rational.test.ft.rational_ft -simplescript true **-datastore** <directory> [**-map**<sharedmapname>] [options] **-record** <script-name>
Record into an existing script, inserting before or after a given line	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [**-insertbefore** <line>] [**-insertafter** <line>] [options] **-map** <sharedmap>] [options] **-record** <script-name>
Compile a script **Note:** You must enable the Java environment before compiling a script with this command. You must also install Java SDK and add the bin directory to the path.	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [options] **-compile** <script-name>
Play back a script by passing command-line arguments <values> to the script **Note:** Ensure that you compile the script before you playback the script from	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-log** <logname> [options] **-playback** <script-name> [**-args** <values>] **Note:** To enable the dynamic find feature for an individual script from the command line, pass [-dynamicfind true] as arguments <values>. To disable, pass [-dynamicfind false].

Table 60.

(continued)

Task	Example
the command prompt for the first time.	If you do not enable or disable the dynamic find feature here, the dynamic find setting on the Dynamic Find Enablement page in the Preferences dialog box, which applies globally to all scripts run in the integrated development environment (IDE), is used.
Play back a script that uses classes from other functional test projects	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-projectpath** <reference-project-path> **-playback** <script-name>
Play back a script that uses other classes for the functional test projects	java **-projectpath** <projectpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-projectpath** <reference-project-path> **-playback** <script-name>
Play back a script that has an associated dataset	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-iterationCount** <iteration value> **-playback** <script-name>
Play back a script and export reports	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-playback** <script-name>**-exportlog** <directory>/<filename without extension>
Record, compile, and play back a script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [options] **-record** <script-name> **-compile -playback** [**-args** <values>] **Note:** To playback your scripts with dynamic VPs, add `-rt.interative true` before `-playback` in the command line.
Construct an empty script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-map** <shared ma pname> [options] **-create** <script-name>
Regenerate the helper file for a script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-helper** <script-name>
Regenerate all helper files for a datastore	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-regenHelpers** <script-name>
Display an object-map file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-display** <object-map filename>
Display a verification-point file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-display** <verification point filename>

Table 60.

(continued)

Task	Example
Edit an object-map file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-edit** <object-map filename>
Edit a verification-point file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-edit** <verification point filename>
Create and edit a shared-object map	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-fromMap** <object-map filename1> **-createMap** <object-map filename2>
Merge a later version of an object map into a current (modified) version of the same map	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-from** <object-map filename1> **-to** <object-map filename2> **-original** <object-map filename1> **-mergeMap**
Compare an actual verification point result to an expected verification point result	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-baseline** <baseline verification point filename> **-compare** <expected verification point filename> <actual verification point filename>
Enable a configured browser, Java™ environment, or Eclipse platform	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-enableName** <browser/Java environment/Eclipse>
Disable all configured browsers, Java™ environments and Eclipse platforms	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-disableall**
Run the Application Configuration Tool	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-appConfig** <application name>
Run the Test Object Inspector	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-inspector**
Run the Object Properties Configuration Tool	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-objectlibraryou**

UI reference

In this section, you will learn all the UI references and its additional information for the Functional Test perspective.

Add Application dialog box

The Add Application dialog box is opened using the **Add** button in the Application Configuration Tool. It is used to add application configurations. When recording tests on your application, it is best to have Rational® Functional TesterHCL OneTest™ UI open the application during recording. This makes playing back the tests more reliable.

You use the Application Configuration Tool to add and configure your own applications for testing and starting with Rational® Functional TesterHCL OneTest™ UI.

Select Application Type

In this first tab, select the type of application you are adding, and click **Next**.

Filename

In this second tab, click **Browse** to locate the application file.

- If it is a Java™ application, select the .class or .jar file of the Java™ application you want to add.
- For an HTML application, select either **Local** or **URL**. If **Local**, browse to an .htm or .html file. If **URL**, enter the URL address.
- For a VB.NET or Windows® application, browse to an executable or batch file.

After selecting any file type, click **Finish**.

Add dynamic test object

You can add dynamic objects to the object map. To add a dynamic object, select an object in the object map by using the **Add Dynamic Test Object** dialog box.

The **Add Dynamic Test Object** dialog box has the following controls:

Anchor to Selected Parent

Select **Anchor to Selected Parent** to make a new object a descendant of its parent. Clear the **Anchor to Selected Parent** check box to use the object as the root.

Object Recognition Properties grid

View lists of recognition properties, values, and weights of the selected object. The value and weights of the object property can be edited.

Finish

Save the changes and close the **Add Dynamic Test Object** dialog box.

Cancel

Cancel all the changes made after the last save operation.

Convert dynamic test object

You can convert an existing mapped object to a dynamic test objects by using **Convert To Dynamic Test Object**.

The **Convert To Dynamic Test Object** dialog box has the following controls:

Select the parent to anchor in the object hierarchy

Select this check box to anchor an object to its parent. Clear **Select the parent to anchor in the object hierarchy** check box if the anchor to the parent does not exist.

Finish

Save the changes and close the **Add Dynamic Test Object** dialog box.

Cancel

Cancel all the changes made after the last save operation.

Add Variable dialog box

You use the Add Variable dialog box to add a new column of variables in a dataset.

Name

Type the name of the new column of variables.



Note: You cannot use spaces in the name.

Type

Type the full class name for the variable. The system String class is the default.

Add

Use to select where you want to add the new column of variables. Click the **Add** arrow to select one of the following options:

Before *XXX*

(where *XXX* is the name of an existing column) Click to place the new column of variables before a particular column.

After *XXX*

(where *XXX* is the name of an existing column) Click to place the new column of variables after a particular column.

To open: Right-click in a dataset, and then click **Add Variable**.

Application configuration tool

This dialog is opened with the **Edit** button in the Start Application dialog box, or by clicking **Configure > Configure Applications for Testing** from Rational® Functional TesterHCL OneTest™ UI. When recording tests on your application, it is best to have Rational® Functional TesterHCL OneTest™ UI start the application during recording. This makes playing back the tests more reliable. Use the Application Configuration Tool to add and configure your own applications for testing and starting with Rational® Functional TesterHCL OneTest™ UI.

How to use this dialog box?

This tool is used to add and edit application configurations. To edit the information on an existing application, click the name of the application in the **Applications** list. To add a new application, click the **Add** button. Whether editing or adding, make your changes, then click **OK** or **Apply** for the changes to be saved.

Applications list

Select the application that you want to edit or view. Its information will then appear to the right of the list. The information fields are described below. If your application is not in the list yet, click **Add** to find and enter it.

Detailed information for *Application*

Contains the following fields:

Name -- This is the logical name of your application. In a Java™ application, this will default to the main class name.

Kind -- The type of application. It will be either "Java™", "HTML," or "executable."

Path -- This is the full path to the application class file. The file path should not contain any spaces. Note that if you type an incorrect path here, the text of the path will turn red in this field to indicate the error. The path should not contain any spaces.

.class/.jar file -- The name of the application file. A .jar file should only be used if the .jar contains a proper manifest specifying the class to be run.

JRE -- This is normally left blank, because it will automatically use the default JRE you have set in the Java™ enabler. (The default is the one listed as "(default)" there.) To use a different JRE for this specific application only, type the name of it in this field, and it will override the default JRE when this application is run. Use the logical name of the JRE, which can also be found in the **Java environments** list in the Java enabler on page 1785.

Classpath -- This is normally the same as the **Path** that is supplied above, or it may be blank. If you need additional classes to run this application, you can enter their paths here and they will get appended to the **Path** listed above.

Args -- This is blank by default. If your application requires command-line arguments, use this field to enter them. They will be passed when the application is started by Rational® Functional TesterHCL OneTest™ UI.

Working Dir -- By default, this is the same as **Path** above -- the full path to the application class file. You only change this if you use a different working directory for this application. Sometimes a single period will appear by default to indicate the default working directory.



Note: When you browse to a new Java™ application using the **Add** button, the **Name**, **Kind**, **Path**, **.class/.jar file**, **Classpath**, and **Working Dir** fields will automatically be filled in for you. The **JRE** and **Args** fields are

optional, and could be filled in by you as described above. When you add a new HTML application, the **Name**, **Kind**, and **Path** fields will be automatically filled in for you.

Add button

Click **Add** to add and configure a new Java™, HTML, or executable/batch file application. For more information on adding applications, see Configuring Applications for Testing on page 796.

Remove button

To remove an application from the **Applications** list, select the application, then click **Remove**.

Run button

Use to run the selected application. This is useful if you want to test the application you are adding or configuring to make sure it runs correctly as you have configured it.

Apply button

If you want to save edits you make in this dialog box before making additional changes, click **Apply** . If you click **Cancel**, any changes you made before you clicked **Apply** will be saved, and changes made after will be canceled.

Font button

Click to change the default font, font style, and font size used throughout the functional test UI.

OK button

You must click **OK** when you are finished to save the additions or edits you made.

Associated Scripts dialog box

You use the Associated Scripts dialog box to view a list of scripts associated with a test object map and to select multiple scripts you want to add test objects to. By default, Rational® Functional TesterHCL OneTest™ UI only selects the active script to add test objects to.

The Associated Scripts dialog box has the following controls:

Select active script(s)

Displays all scripts associated with the current test object map. To select multiple scripts to add test objects to, press and hold **Ctrl** while you click the names of the scripts.

OK

Closes the display. If you selected one or more scripts, Rational® Functional TesterHCL OneTest™ UI places the selected object in the Script Explorer for each script. If you use Source control, Rational® Functional TesterHCL OneTest™ UI automatically checks out the scripts unreserved and leaves them checked out when you add test objects to them.

Rational® Functional TesterHCL OneTest™ UI changes the **Test Object > Add to Script** menu item to **Test Object > Add to Multiple Scripts** or the tooltip for the **Test Object: Add to Script** button to **Add to Multiple Scripts** to indicate that multiple scripts have been selected and will be affected by the command.

To open: On the Test Object Map toolbar, click the **Test Object: Associated Scripts** button or in the Test Object Map menu, click **Test Object > Associated Scripts**.

Bookmarks view

The Bookmarks View displays a list of markers that point to a specific place in the Workbench.

The Bookmarks View has the following tools:

Delete Bookmark

Deletes the selected bookmark.

Go to file

Opens an editor containing the file associated with the bookmark.

Refer to the online *Workbench User Guide* for more information.

To open: In the product menu, click **Window > Show View > Bookmarks**.

Browser enablement diagnostic tool

The Browser Enablement Diagnostic Tool is used to diagnose problems you might have with enabling your browser for HTML testing. The tool will diagnose the enablement problem and report how to solve the problem.

About this task

Use the diagnostic tool if you suspect that HTML is not being tested properly. If you are trying to record against an HTML application, and nothing shows up in the Recording Monitor, the browser is probably not enabled properly. It might mean that the Java™ plug-in of your browser is not enabled. If that is the case, the diagnostic tool will tell you how to enable the browser. The tool offers quick and simple directions to solve any problem it finds.

To run the tool:

1. Open the Rational® Functional TesterHCL OneTest™ UI Enabler by clicking **Configure > Enable Environments for Testing**.
2. Click the **Web Browsers** tab.
3. Click the **Test** button. The Browser Enablement Diagnostic Tool opens.
4. Click the **Run Diagnostic Tests** button.

Results

About this task

The **Results** page tells you whether the test passed or failed. If the test failed, this page will also list the problem.

Problem and solution

About this task

The **Problem and Solution** page will list the problem and explain how to solve it. Follow the instructions listed there and close the tool. If you were in the process of recording a script when you ran the tool, stop recording the script and start over. The recording should then work against an HTML application.

Details (Advanced)

About this task

The **Details** page list additional information about your environments. The **Java Enabled** field indicates whether Java™ is enabled in your browser. The **JVM Information** field lists information about your JVM. The **General Enablement Information** field lists Java™ and HTML domain information.

Call Script tab: Script Support Functions dialog box

You use the Call Script tab to insert a callScript command into your Functional Test script.

The **Call Script** tab has the following controls:

Script Name

Lists all the scripts in the current project.

Insert Code

Inserts the `callScript(" scriptname")` code in the current script at the cursor location, where `scriptname` is the name you selected in the **Script Name** field.

dataset Iterator Count

Determines how many times a test script runs when you play back the test script. Specify the count according to the number of records you have in the dataset. Type or select the number of records in the dataset, or select **Iterate Until Done** to access all records in the dataset. For a call script, you can select **Use Current Record** to use the same record across the call script.



Note: You can also insert a callScript command from the Functional Test Projects view or script it manually.

To open: If recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar and click the **Call Script** tab. If editing, click the **Insert Recording into Active Functional Test Script** button  on the product toolbar, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar, and click the **Call Script** tab.

Edit Variable dialog box

You can use the Edit Variable dialog box to change the name of a column of variables, the type of variable, or the position of the variable in a dataset.

Name

Type a new name for the column of variables.

Note: You cannot use spaces in the name.

Type

Use to change the default class of the variable. Type the full class name for the variable. The systems String class is the default, if not explicitly specified.

Move

Use to move a column of variables in a dataset. Click the **Move** arrow to select one of the following options:

Before *XXX*

(where *XXX* is the name of an existing column) Click to move a column of variables before a particular column.

After *XXX*

(where *XXX* is the name of an existing column) Click to move a column of variables after a particular column.

To open: Right-click in a dataset, and then click **Edit Variable**.

Choose Test Object to Update page of the Update Recognition Properties wizard

You use the Choose Test Object to Update page of the Update Recognition Properties wizard to select the test object for which you want to update recognition properties.

The Choose Test Object to Update page has the following controls:

<>

Displays an icon to indicate how well the object property in the test object map matches the property of the object in the application-under-test. It is a graphic representation of the **Score** column and uses the following icons:

Match indicates that the properties were an exact match with a recognition score of 100.

Partial indicates that the properties were an acceptable match based on the **Recognition Level** on the Standard ScriptAssure(TM) page on page 1441 or the **Maximum acceptable recognition score** and the **Last chance recognition score** on the Advanced ScriptAssure(TM) page on page 840.

No Match indicates that the properties did not match with a score of 0.

Okay indicates that the test objects were an acceptable match.

Very Bad indicates that the test objects were not an acceptable match.

Mapped Test Object

Displays the test object you selected and the parent hierarchy of the test object. Expand the test object name to display the properties for the test object and the parent.

Score

Indicates how well the object property in the test object map matches the property of the object in the application under test:

Match

Indicates that the properties were an exact match with a recognition score of 100.

Partial *n*

Indicates that the properties were an acceptable match based on the **Recognition Level** on the Standard ScriptAssure(TM) page on page 1441 or the **Maximum acceptable recognition score** and the **Last chance recognition score** on the Advanced ScriptAssure(TM) page on page 840. ***n*** is the recognition score.

No Match

Indicates that the properties did not match with a score of 0.

Okay

Indicates that the test objects were an acceptable match.

Very Bad

Indicates that the test objects were not an acceptable match.

Back

Displays the Select an Object page of the Update Recognition Properties page, which enables you to choose another test object.

Next

Displays the Update Test Object Recognition Properties page on page 1842, which enables you to confirm the updated recognition property.

To open: Select an object in the test object map, start the application, and click **Test Object > Update Recognition Properties**.

Clipboard tab: Script Support Functions dialog box

Use this tab to insert a system clipboard command into a functional test script.

The Clipboard tab has the following tabs:

- **Verification Point**: Use to insert a verification point test command against the active content in the system clipboard into a functional test script.
- **Assign Text**: Use to insert the text from the system clipboard to a variable in a functional test script.
- **Set Text**: Use to update the contents of the system clipboard to a required value.

Verification Point tab

The Verification Point tab has the following controls:

- **Convert Value to Regular Expression**: Use to convert the system clipboard text to a regular expression pattern to be matched at run time against the system clipboard contents.
- **Undo Regular Expression**: Use to revert the regular expression pattern back to the active system clipboard contents at the time the Script Support Functions dialog box was requested.
- **Evaluate Regular Expression**: Use to evaluate the current pattern against the active system clipboard contents at the time the Script Support Functions dialog box was requested.
- **VP Name**: Name used for the clipboard verification point in the script. This name must be unique relative to the associated script.
- **Insert Code**: Inserts the clipboard verification point command into the active functional test script.

The text displayed initially represents the active system clipboard content at the time the Script Support Functions dialog box was requested. If regular expression pattern matching support is requested, this value becomes the pattern matched against the active system clipboard contents during playback.

Assign Text tab

The Assign Text tab has the following controls:

- **Variable Name**: Defines the variable name used in the functional test script.

  Note: This name must be a valid variable name for the target script language. Spaces and special characters are not allowed.

- **Precede variable assignment with type declaration**: Use to precede the variable name with a String type declaration.
- **Insert Code**: Inserts the command to assign the system clipboard content into a local variable in a functional test script.

Set Text tab

The Set Text tab has the following controls:

- **Set clipboard text to the following value**: Sets the content for the system clipboard with a specific value.
- **Insert Code**: Inserts the command to set the content of the system clipboard to the supplied value into a functional test script

Comment tab: Script Support Functions dialog box

You use the Comment tab to insert a comment into a Functional Test script.

The **Comment** tab has the following controls:

Comment to add to the script

Enter text for the comment.

Note: Rational® Functional TesterHCL OneTest™ UI does not automatically wrap the text. Put returns after each line.

Insert Code

Inserts the text with the appropriate comment delimiter (//) preceding each line.

To open: When recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar and click the **Comment** tab.

Configure Handling of Unexpected Windows dialog box

Use the Configure Handling of Unexpected Windows dialog box to configure actions to be performed when unexpected windows open during script playback. Configuring these actions helps to ensure that scripts play back smoothly without interruptions.

The Configure Handling of Unexpected Windows dialog box contains the following controls:

Select the Test Domain

Lists the domains for which you can configure unexpected windows.

Perform close action for 'non-configured' windows

If selected, all unexpected windows in the selected domain that have not been configured are closed automatically.

Unexpected Window Title

Lists all unexpected windows that can be configured for the selected domain.

Add Window

Adds a window, if the required window is not listed for the selected domain.

Remove Window

Removes a window from the list of unexpected windows for the selected domain.

Select Action

Lists the actions that can be configured for an unexpected window. An unexpected window can be configured either to be closed automatically, or have a specific action performed on it, depending on by additional recognition properties that you can define.

Configure action objects properties to perform selected action

To add object recognition properties that will help identify the action to be performed on a specific control on the unexpected window. This field is unavailable if the Close action was selected in the **Select Action** list. You can specify a Property Name and Property Value for the control.

Add Property

Adds an object recognition property for the control on the unexpected window. You can also use the Test Object Inspector to get properties for the control. Open the unexpected window and the Test Object Inspector. Move the cursor over the window to get the title. Move the cursor over the specific control, to get its property name and value. Refer to Displaying test object information on page 885 for instructions to use the Test Object Inspector.

Remove Property

Removes an object recognition property for the control on the unexpected window.

Finish

Saves the changes and closes the Configure Handling of Unexpected Windows dialog box.

Cancel

Cancels all the changes made in the Configure Handling of Unexpected Windows dialog box after you last saved.

Apply

Saves the changes without closing the Configure Handling of Unexpected Windows dialog box.

Connect to a Functional Test project dialog box

You can use the Connect to a Functional Test Project dialog box to connect to an existing Functional Test project.

Functional Test Project location path

Type the path for the new Functional Test project or click **Browse** to select a path of an existing Functional Test project. Rational® Functional TesterHCL OneTest™ UI automatically enters the folder name in the **Project name** field.

Project name

Type a name to represent the project or an alias for the project. This name must be unique.

To open: Click **File > Connect to a Functional Test project**.

Console view

The Console View displays output from the script or application, for example, System.out.print statements or unhandled exceptions.

To erase all text from the Console View, click the **Clear Output**  button.

To open: In the Functional Test Perspective, click the **Console** tab.

Copy Test Objects to New Test Object Map wizard page

You use the Copy Test Objects to New Test Object Map wizard page to create a new test object map without any objects or to base the map on another object map.

The Copy Test Objects to New Test Object Map wizard page has the following controls:

Don't copy any Test Objects

Creates a new, empty object map.

Select Test Object Maps and scripts to copy Test Objects from

Creates a new object map, using one or more maps you select in the list as a template. Selecting this option displays a list of all the test assets in the project.

Connect selected scripts with new Test Object Map

Associates the new map to the selected scripts immediately. This option is available only if a script is selected in the list.

Select this Test Object Map as default choice for new scripts (Rational Functional TesterHCL OneTest UI, Microsoft Visual Studio .NET Integration only)

Automatically highlights the script name and indicates the test object map as the default in the Solution Explorer.

Back

Returns to the first page of the Create New Test Object Map wizard on page 1747.

Finish

Creates and displays the new object map based on the criteria you specified. Rational® Functional TesterHCL OneTest™ UI updates any scripts you selected to reference the new test object map. If you added the script to Source Control, Rational® Functional TesterHCL OneTest™ UI checks the script out unreserved and leaves it checked out.

Cancel

Closes the Copy Test Objects to New Object Map wizard page without creating a new object map.

To open: In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, when creating a new test object map, click **Next** in the Create New Test Object Map wizard. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, click **Open** in the Add New Item dialog box for creating a new test object map.

Create a New Functional Test Folder dialog box

You use the Create a New Functional Test Folder dialog box to add a new Functional Test folder to the Projects view.

The Create a New Functional Test Folder dialog box has the following controls:

Enter or select the folder

Either enter the appropriate path to the folder you want to create or use the navigation tools (**Home**  , **Back**  , and **Go Into** ) to select the path in the selected project.

Test folder name

Enter the name for the new folder.

Finish

Creates the new folder in the project you selected.

Cancel

Closes the Create a New Functional Test Folder dialog box without creating the new folder.

To open: In the product menu, select **File > New > Test Folder**. In the product toolbar, select the **Create a Test Folder** button  . In the Projects view, select a file, right-click and select **New Test Folder**.

Create a New Project or Connect to an Existing Project dialog box

You can use the Create a New Project or Connect to an Existing Project dialog box to create a new project or connect to an existing project to record a new script. This dialog box opens when no projects are available and you attempt to record a script.

The Create a New Project or Connect to an Existing Project dialog box has the following controls:

Create a new Functional Test project

Opens the Create New Functional Test Project dialog box.

Connect to an existing Functional Test project

Opens the Connect to a Functional Test Project dialog box.

To open: With no projects in Functional Test, click the **Record a Functional Test Script** button  on the product toolbar.

Create a test dataset dialog box

You can use the Create a Test dataset dialog box to create a new empty test dataset.

Enter or select the folder

Either type the appropriate path to the folder you want to use or use the navigation tools (Home  , Back  , and Go Into ) to select the path.

dataset name

Type the name you want to use for a new dataset.

To open: Click **File > New > Test dataset** from the product menu or click **Create a Test dataset** ( ), on the product toolbar.

Create a New Test Object Map wizard

You use the Create a New Test Object Map wizard to create a customized, shared test object map that you can associate with scripts. Shared test object maps have the extension .rftmap.

The first page of the Create a New Test Object Map wizard has the following controls:

Enter or select the folder

Use the navigation tools (**Home** , **Back** , and **Go Into**) to select the appropriate path to the project you want to use.

Map Name

Enter the name you want to use for the new test object map. Rational® Functional TesterHCL OneTest™ UI appends the .rftmap extension to the name.

Set this Test Object Map as default choice for new scripts

Automatically highlights the script name and indicates the test object map as the default in the Projects view.

Next

Displays the Copy Test Objects to New Test Object Map page on page 1745, which enables you to create a new empty test object map or to base the map on one or more other object maps for an existing script.

Finish

Creates a default test object map.

Cancel

Closes the Create a New Map wizard without creating a new test object map.

To open: In the product menu, click **File > New> Test Object Map**. In the product toolbar, click the **Create a Test Object Map** button .

Create an empty Functional Test script dialog box

You use the Create an empty Functional Test script dialog box to write an empty Functional Test script without recording. As an alternative to recording, you can then enter the Java™ code manually.

The Create an empty Functional Test script dialog box has the following controls:

Enter or select the folder

Either enter the appropriate path to the folder you want to use or use the navigation tools (**Home** , **Back** , and **Go Into**) to select the path.

Script name

Enter the name you want to use for the new Functional Test script. **Tip:** Use Java™ file naming conventions.

Next

Displays the Select Test Object Map wizard page on page 1834, which enables you to choose a private or a shared object map to use with the new Functional Test script.

Finish

Creates a new Functional Test script, using the default test object map.

Cancel

Closes the Create a New Functional Test Script dialog box without creating the new script.

To open: Click the **Create an Empty Functional Test Script** button on the product toolbar or click **File> New> Empty Functional Test Script** on the product menu.

Create a Functional Test Project dialog box

You can use the Create a Functional Test Project dialog box to create a new functional test project.

Note: If you clicked the **New** button on the product toolbar to create a new Functional Test project, the New dialog box appears. The **Select a Wizard** page displays a list of project wizards.

Project name

The name of the new Functional Test project. Project names cannot contain the following characters: \ / : * ? " <> | ( ) or a space. If your project is on a UNIX® computer, do not embed spaces in the project name.

Project location

The path of the new project. You can type this path or click **Browse** to select a path.

Note:

- If this check box is unavailable, you may not have the appropriate privileges set up.

To open: Click the **Create a Functional Test Project** button on the product toolbar and on the Functional Test menu, click **File> New> Functional Test Project**.

Create Script Helper Superclass dialog box

If you are an advanced user, use the Create Script Helper Superclass dialog box to create your own helper superclass in a new script, which extends `RationalTestScript` and adds additional methods or overrides the methods from `RationalTestScript`.

The Create Script Helper Superclass dialog box has the following controls:

Enter or select the folder -- Either enter the appropriate path to the folder you want to create or use the navigation tools (**Home**  , **Back** , and **Go Into**) to select the path in the selected project.

Project name -- Displays the project names in the current folder.

Script name -- Enter the name for the class.

Finish -- Creates a new script in the Java™ Editor that you can use to manually enter Java™ code for the helper superclass.

Cancel-- Closes the Create Script Helper Superclass dialog box without creating the new script.

To open: In the product menu, click **File > New > Helper Superclass** or click the **View Menu** button  next to the **New** button on the product toolbar and click **Helper Superclass**.

dataset Literal Substitution dialog box

You can use the dataset Literal Substitution dialog box to find or replace literal values in a test script with a dataset reference (an associated dataset). You can set the options in this dialog box to find and replace all, number, string, or boolean literals in a script with a dataset reference.

You can also add a literal from a script to a dataset. If you do not use an existing dataset variable, Rational® Functional TesterHCL OneTest™ UI uses the same literal values (the values that Rational® Functional TesterHCL OneTest™ UI captured when you recorded the test script) each time you run the script.

Literal

Displays the name of a selected literal in a script.

dataset Variable

Type the name of the dataset variable with which you want to replace a literal or click the **dataset Variable** arrow to select the existing dataset variable that you want the script to reference. The default type for a new dataset column is string. You can change the variable type on page 1739 later.

Direction

Select the direction that you want to move through a script. This setting works with the**Literal Type** setting. Click **Find** or **Replace** to move either forward or backward through a script starting from the insertion point of your cursor.

Forward

Click to move forward to the next literal in the script from the insertion point.

Backward

Click to move backward to the next literal in the script from the insertion point.

Literal Type

Click to set the type of literal that you want to find; either all, number, string, or boolean literals in a script. A literal is a letter or symbol that stands for itself as opposed to a feature, function, or entity associated with the literal in a programming language: $ can be a symbol that refers to the end of a line, but as a literal, it is a dollar sign. The **Literal Type** setting works with the **Direction** setting. Click **Find** or **Replace** to move to the next type of literal through a script starting from the insertion point of your cursor.

All

Click to find all literals in a script.

Numbers

Click to find number literals in a script. A number includes integers (a whole number, not a fractional number, that can be positive, negative, or zero) or floating numbers (positive and negative decimal numbers).

Strings

Click to find string literals in a script. A string stores alphanumeric values such as name, city, or state.

Booleans

Click to find boolean literals in a script. Any use of the boolean literals true or false are flagged for substitution.

Add Literal to dataset

Click to add a selected literal to a dataset variable when you click **Replace**. The selected literal is added to the dataset in the variable that you specify in **dataset Variable**. A dataset reference replaces the selected literal when you click **Replace**.

Find

Click to find the next literal in a script. The **Find** setting works with the **Direction** and **Literal Type** settings.

Replace

Click to replace the selected literal with a dataset reference. The dataset reference is made to the dataset variable you specify in **dataset Variable**. When you click **Add literal to dataset**, Rational® Functional TesterHCL OneTest™ UI adds a selected literal to the dataset variable you specify in **dataset Variable**. Automatically moves to the next literal in the test script after replacing a literal. The **Replace** setting works with the **Direction** and **Literal Type** settings.

Close

Click to close the dialog box without finding a literal or replacing a literal with a dataset reference.

To open: Click **Script> Find Literals and Replace with dataset Reference**.

dataset Reference Converter dialog box

You can use the dataset Reference Converter dialog box to convert a literal in a verification point into a dataset variable.

dataset Variable

Click the **dataset Variable** arrow to select the variable you want the verification point to reference in the dataset or type a new name for the new dataset variable reference.

Add value to new record in dataset

Select this check box to add the value of the verification point to a new record (row) in the dataset table. Clear this check box to add the verification point value to all records in the dataset.

To open: Create a verification point. From the **Verification Point and Action Wizard**, for a data verification point, click **Convert Value to dataset Reference** (). For a properties verification point, click a property for which you want a dataset reference, and right-click the property you want, and click **Convert Value to dataset Reference**.

Define Find Filter Name dialog box

The Define Find Filter Name dialog box is the third step in creating and editing find criteria, which enables you to use filters to find objects in a test object map. You use the dialog box to name the set of find criteria.

The Define Find Filter Name dialog box has the following controls:

Filter Name

Enter the name you want to use for this set of find criteria. This is the name Functional Test displays in the **Find Filter Names** field of the Set Active Find Criteria dialog box.

Tip: When editing, changing the **Filter Name** copies the set of criteria, creating a new set of criteria and leaving the old set intact.

Back

Returns to the Define Find Filter Relationships dialog box on page 1753.

Next

Re-displays the Set Active Find Criteria dialog box on page 1835, which enables you to run the filter you just created, define a new filter, or edit an existing filter.

Finish

Saves any changes, closes the Define Find Filter Name dialog box, and redisplays the Set Active Find Criteria dialog box, which enables you to run the filter you just created or edited, define a new filter, or edit an existing filter.

Cancel

Closes the Define Find Filter Name dialog box without saving any changes.

To open: When defining or editing find criteria, click **Next** in the Define Find Filter Relationships dialog box.

Define Find Filter Properties dialog box

The Define Find Filter Properties dialog box is the first step in creating and editing find criteria, which enables you to use filters to find objects in a test object map. You use this dialog box to specify properties for the new search criteria or to edit properties for an existing one.

The Define Find Filter Properties dialog box has the following controls:

Properties

Lists all the available properties for all the objects in the current test object map.

Filter

Lists the properties you selected and their relationships.

Note: If you add more than one property to the **Filter** field, Rational® Functional TesterHCL OneTest™ UI groups the objects under an **AND** folder (the default). You can right-click the **AND** folder to change it to **OR**. Right-clicking objects in the **Filter** field enables you to cut, copy, paste, or delete them. To group objects, press **Ctrl** and click the objects you want to include, right-click, and select **Group**. To ungroup objects, right-click the group folder and select **Ungroup**.

>>>

Adds the selected property to the **Filter** field.

<<<

Removes the selected property from the **Filter** field.

Back

Returns to the Set Active Find Criteria dialog box on page 1835.

Next

Continues the procedure by displaying the Define Find Filter Relationships dialog box on page 1753.

Finish

Available when editing existing find criteria, saves any changes, closes the Define Find Filter Properties dialog box, and redisplays the Set Active Find Criteria dialog box, which enables you to run the filter you just created, define a new filter, or edit an existing filter.

Cancel

Closes the Define Find Filter Properties dialog box without saving any changes.

To open: When defining or editing find criteria, click **Create** in the Set Active Find Criteria dialog box.

Define Find Filter Relationships dialog box

The Define Find Filter Relationships dialog box is the second step in creating and editing find criteria, which enables you to use filters to find objects in a test object map. You use this dialog box to define the relationship between the properties you added to the find filter.

The Define Find Filter Relationships dialog box has the following controls:

Filter

Lists all the groups of filter properties you specified in the Define Find Filter Properties dialog box. If necessary, click **<Back** to make any additions or edits to the list.



Note: You can right-click a group folder to change it to **AND** or **OR**. To group objects, press Ctrl and click the objects you want to include, right-click, and select **Group**. To ungroup objects, right-click the group folder and select **Ungroup**. Right-clicking objects in the **Filter** field enables you to cut, copy, paste, or delete them.

Relationship

The controls in this section change based on whether you click a group or a property in the **Filter** box.

AND

When selected, changes the highlighted group to an **AND**; all the property relationships in the group must resolve to true in order for Rational® Functional TesterHCL OneTest™ UI to find the object in the test object map.

OR

When selected, changes the highlighted group to an **OR**; at least one property relationship in the group must resolve to true in order for Rational® Functional TesterHCL OneTest™ UI to find the object in the test object map.

Operator

Enables you to indicate whether the **Value: Exists** (the default), **IsNull**, or is **Equal**.

NOT Relationship

When selected, Rational® Functional TesterHCL OneTest™ UI searches for objects that do not contain the specified relationship and value.

Value

For the specified property, enter a value you want Rational® Functional TesterHCL OneTest™ UI to search for.

Convert Value to Regular Expression

Available when **Operator** is **Equal**, enables you to specify a regular expression in the **Value** field. For information, see Replacing an Exact-Match Property with a Pattern.

Convert Value to Numeric Range

Available when **Operator** is **Equal**, enables you to specify a numeric range in the **Value** field. For information, see Replacing an Exact-Match Property with a Pattern. Clicking the **NR** notation in the **Value** field displays fields that enable you to specify the numeric range parameters. This button is only enabled if the value is of a numeric type.

Back

Returns to the Define Find Filter Properties on page 1752 dialog box.

Next

Advances to the Define Find Filter Name on page 1751 dialog box.

Finish

Available when editing existing find criteria, saves any changes and closes the Define Find Filter Relationships dialog box.

Cancel

Closes the Define Find Filter Relationships dialog box without saving any changes you have made.

To open: When defining or editing find criteria, click **Next** in the Define Find Filter Properties dialog box.

Delete All Not Used Test Objects dialog box

You use the Delete All Not Used Test Objects dialog box to find all the test objects that do not have references in the scripts associated with the shared test object map and selectively delete them.

The Delete All Not Used Test Objects dialog box has the following controls:

Find: First

Moves to the first test object in the list.

Find: Previous

Moves to the previous test object in the list.

Find: Next

Moves to the next test object in the list.

Find: Last

Moves to the last test object in the list.

Test objects highlighted in red are not used in the scripts associated with the test object map. The check boxes fir these objects are selected and the objects are deleted when you click **OK**. Clear the check box of any test object that you do not want to delete.

Children of parents marked for deletion are marked for deletion. When parents are deleted, all children are deleted also. To keep one or more children, clear the parent check box and clear the check boxes of the children you do not want to delete.

The lower pane contains property sets, which provide information about the selected object. There are two property set tabs:

- **Recognition**
- **Administrative**

The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI. The **Administrative** tab displays internal administrative data of the object. These properties are used to manage and describe the test object.

Delete Test Object dialog box

This is the first page of the Delete Test Object wizard, which enables you to confirm that you want to delete the selected object from the test object map.

The Delete Test Object dialog box has the following controls and information:

The Delete Test Object dialog box lists the Administrative and Recognition properties for the object you are about to delete.

Total number of test objects deleted

Displays the number of test objects associated with the selected object that will also be deleted.

Next

Displays the second page of the wizard on page 1756, which lists all the scripts that reference the object and that will be affected by the deletion.

Finish

Deletes the object from the test object map and deletes all references to the object.

Cancel

Closes the Delete Test Object dialog box without deleting the object.

To open: Select a test object and click **Edit > Delete** or click the **Delete** button  on the Object Map toolbar. You can also right-click a test object and click **Delete**.

Delete Test Object dialog box -- page 2

This is the second page of the Delete Test Object wizard, which enables you to confirm that you want to delete the selected test object.

It has the following controls and information:

The Delete Test Object dialog box lists all the scripts that reference the object and that will be affected by the deletion.

Back

Returns to the first page of the Delete Test Object wizard on page 1755, which displays the number of test objects associated with the selected object that will also be deleted and lists the Administrative and Recognition properties for the object you are about to delete.

Finish

Deletes the object from the object map and deletes all references to the object.

Cancel

Closes the Delete Test Object dialog box without deleting the object.

To open: Click the **Next** button on the first page of the Delete Test Object wizard.

Eclipse Platforms tab of the Enable Environments dialog box

You must enable the Eclipse platform for testing before recording scripts for testing Eclipse-based applications.

Eclipse Platform tab

This tab has the following fields:

Eclipse Platforms

Displays the list of Eclipse platforms that the enabler locates on your hard disk drive(s). This list is populated when you click the **Search** button. After the name, the enabler indicates in parentheses whether that platform is currently enabled.

Select All

To select all the platforms that are listed in the **Eclipse Platforms** list. This is useful if you want to enable or disable all the platforms. To clear them all, click any of the individual platform

Search

Click **Search** to specify the search options for Rational® Functional TesterHCL OneTest™ UI to search the location in the computer for Eclipse platforms.

Add

Click **Add** to locate platforms individually. The selected platform is added to the **Eclipse Platforms** list. The main use of **Add** would be if you only want to browse to one specific platform instead of searching. You can also use the **Search In** option in the Search for Eclipse Platform dialog box to locate an individual platform.

Remove

If you want to remove a platform from the list, select it and click **Remove**.

Enable

Use this option to enable the selected platforms for testing with Rational® Functional TesterHCL OneTest™ UI. Select the platform from the list and click **Enable**.

Disable

Use this option to disable selected platforms for testing with Rational® Functional TesterHCL OneTest™ UI.

Detailed Information for *Platform*

Contains the following fields:

Name

This is the name of the directory that contains the Eclipse directory. You can edit this name.

Path

This is the full path to the platform installation.

Gef Support

Select the Gef Support check box to use Rational® Functional TesterHCL OneTest™ UI to test the functionality of GEF objects that are implemented using standard and non-standard GEF editors.

Edit Test Object Appearance dialog box

You use the Edit Test Object Appearance dialog box to customize the text for test objects displayed in the hierarchy tree of the test object map.

You can add or delete any of the properties of the test object using the following format for each property you want to add:

`%map: property%:`

where *property* is the name of the property as it appears in the property sheet for the object.

For example, you can add the descriptive name to the display.

`%map:#name%:`

Use a pair of carets (^) to indicate properties that are not common to all objects or those that return a null value. This ensures that gaps or punctuation marks do not appear in the descriptions of objects that do not have the property.

For example, the following ensures that no gaps or punctuation marks appear in the test object map text for objects that do not have a descriptive name:

```
^%map:#name%:^
```

To open: From the object map menu, click **Preferences > Test Object Appearance on Tree**.

Edit Variable dialog box

You can use the Edit Variable dialog box to change the name of a column of variables, the type of variable, or the position of the variable in a dataset.

Name

Type a new name for the column of variables.

Note: You cannot use spaces in the name.

Type

Use to change the default class of the variable. Type the full class name for the variable. The systems String class is the default, if not explicitly specified.

Move

Use to move a column of variables in a dataset. Click the **Move** arrow to select one of the following options:

Before *XXX*

(where *XXX* is the name of an existing column) Click to move a column of variables before a particular column.

After *XXX*

(where *XXX* is the name of an existing column) Click to move a column of variables after a particular column.

To open: Right-click in a dataset, and then click **Edit Variable**.

Browser enablement diagnostic tool

The Browser Enablement Diagnostic Tool is used to diagnose problems you might have with enabling your browser for HTML testing. The tool will diagnose the enablement problem and report how to solve the problem.

About this task

Use the diagnostic tool if you suspect that HTML is not being tested properly. If you are trying to record against an HTML application, and nothing shows up in the Recording Monitor, the browser is probably not enabled properly. It

might mean that the Java™ plug-in of your browser is not enabled. If that is the case, the diagnostic tool will tell you how to enable the browser. The tool offers quick and simple directions to solve any problem it finds.

To run the tool:

1. Open the Rational® Functional TesterHCL OneTest™ UI Enabler by clicking **Configure > Enable Environments for Testing**.
2. Click the **Web Browsers** tab.
3. Click the **Test** button. The Browser Enablement Diagnostic Tool opens.
4. Click the **Run Diagnostic Tests** button.

Results

About this task

The **Results** page tells you whether the test passed or failed. If the test failed, this page will also list the problem.

Problem and solution

About this task

The **Problem and Solution** page will list the problem and explain how to solve it. Follow the instructions listed there and close the tool. If you were in the process of recording a script when you ran the tool, stop recording the script and start over. The recording should then work against an HTML application.

Details (Advanced)

About this task

The **Details** page list additional information about your environments. The **Java Enabled** field indicates whether Java™ is enabled in your browser. The **JVM Information** field lists information about your JVM. The **General Enablement Information** field lists Java™ and HTML domain information.

Export dialog box

You can use the Export dialog box to export a private or public dataset to a .csv file.

You can export a dataset to:

- Add data to a dataset using a spreadsheet application. For example, you can export a dataset to a .csv file and then use a spreadsheet application to add more data. After you finish adding data, you can then import the .csv file into a Functional Test dataset.
- Use a dataset in a different project. For example, you can export a dataset to a .csv file and then import the data into a new dataset in a different Functional Test project.

For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration:

Select a public dataset to export

If you select a public dataset to export from the Functional Test Projects view, use the navigation tools (Home  , Back  , and Go Into ) to select a public dataset to export.

File

Type the path and file name of the location of the .csv file into which you are exporting the dataset.

Browse

Click to specify a destination directory for the .csv file.

Field Separator

Type the field separator character that you want to use in the .csv file.

To open:

ForRational® Functional TesterHCL OneTest™ UI, Eclipse Integration -- For a public dataset, in the Functional Test Projects window, right-click any public dataset, and then click **Export**. Click **Functional Test dataset to CSV File**. Click **Next**.

For Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration -- For a public dataset, in the Solution Explorer, select the public dataset you want to export, and then right-click **Export**.

For a private dataset, open the test script associated with the private dataset that you want to export, in the Script Explorer select the dataset you want to export, and then right-click **Export**.

Find and modify dialog box

You use the Find and Modify dialog box to search for test objects in a test object map based on a property, a value, or either. After the object is found, you can make several types of modifications to its properties and values.

The Find and Modify dialog box has the following controls:

Find criteria:

Quick Find

Select to search a test object map for an object based on the property or value you enter.

Find

Enter the string you want to search for.

Match Case

Select to find only the text that matches the case of the string you entered in the **Find** field.

property

Select to search only properties for the text you entered.

value

Select to search only values for the text you entered.

either

Select to search properties or values for the text you entered.

Find By Filter

Select a filter to use for searching the test object map, to create or edit a filter, and to delete an existing filter.

Filter

Lists the names of all the active find criteria available. The default is Test Object is New, which searches the test object map for all New objects.

Create

Opens the Define Find Filter Properties on page 1752 dialog box, which enables you to specify properties for a new set of find criteria.

Edit

Opens the Define Find Filter Properties on page 1752 dialog box, which enables you to change properties for the selected find criteria.

Delete

Deletes the selected find filter from the list.

Modify actions:

Action

Select the modification to make to the test object: **Add Property**, **Remove Property**, **Change Value**, **Change Weight**, or **Change Value and Weight**.

Property

Select the property you want to add, delete, or modify.

Value

Add the value of a new property or edit the value of an existing property.

Weight

Enter a number from 0 to 100 to indicate the importance of the property when searching for it in an application-under-test. For more information about test object weight, see Using ScriptAssure(TM) on page 1441. Read-only properties do not have a weight.

Next

Finds the next test object in the map that meets the find criterion.

Modify

Makes the changes you specified in the **Modify actions** section and moves to the next test object that matches the find criterion.

Modify All

Applies the changes in the **Modify actions** section to all the test objects in the object map that match the criterion.

Rational® Functional TesterHCL OneTest™ UI menus

This topic describes all the options and commands on the Rational® Functional TesterHCL OneTest™ UI menu.

Rational® Functional TesterHCL OneTest™ UI has the following menus:

File on page 1762

Edit on page 1764

Source on page 1765

Refactor on page 1765

Navigate on page 1765

Search on page 1766

Project on page 1766

Script on page 1766

Configure on page 1766

Run on page 1767

Window on page 1767

Help on page 1767

File menu

The **File** menu has the following commands:

New -- Displays the appropriate dialog box that enables you to create a new project, Functional Test project on page 858, record a Functional Test script on page 877, Functional Test empty script on page 879, test folder on page 862, test dataset on page 943, test object map, helper superclass on page 889, or to use all wizards other than project wizards.

Connect to a Functional Test project -- Displays the Connect to a Functional Test project dialog box, which enables you to specify the project you want to use and its location. You must first connect to an existing Functional Test project on page 859 before you can use it.

Open File -- Displays the Open File dialog box, which enables you to open a file that is not part of a Functional Test project.

Close -- Closes the script currently displayed in the Java™ Editor.

Close All -- Closes all the scripts that are open in the Java™ Editor.

Save *script* -- Saves any changes you made to the script currently displayed in the Java™ Editor.

Save Script " *script* " As -- Displays the Save Script As dialog box, which enables you to save the current Functional Test script under another name on page 898.

Save As -- Displays the Save As dialog box, which enables you to save the current file under another name and location on page 898.



Note: Do not use this option to save scripts.

Save All -- Saves all the scripts and files currently open.

Revert -- Reverts the contents of the script back to the previous saved version.

Move -- Enables you to move the selected editor or tab group.

Rename -- Displays the Rename dialog box, which enables you to save the script with a different name.

Refresh -- Refreshes the contents of the currently active script to reflect changes.

Convert Line Delimiters To -- Enables you to see the line delimiters in default view (Windows), or convert to either Unix or MacOS 9.

Print -- Displays the Print dialog box, which enables you to print the currently active script.

Switch Workspace -- Displays the Select a workspace dialog box, which enables you to store projects in a different directory only for the current session.

Restart

Import -- Displays the Import wizard, which enables you to copy files from an external source into projects. For information, see the online *Workbench User Guide*.

Note: Functional Test supports two import types: Functional Test project items and Functional Test datasets.

Export -- Displays the Export Wizard, which enables you to export resources from projects to an external source. For information, see the online *Workbench User Guide*. *Note:* Functional Test supports two export types: Functional Test project items and Functional Test datasets.

Properties -- Displays the Rational Functional TesterHCL OneTest UI Project Properties page on page 1770 page if you have selected a Functional Test project, or the Rational Functional TesterHCL OneTest UI Script Properties page on page 1773, if you have selected a script.

Project filenames -- Displays the names of projects you have most recently used.

Exit -- Closes Rational® Functional TesterHCL OneTest™ UI.

Edit menu

The **Edit** menu has the following options:

Undo -- Cancels the last edit you made, if possible. Some edits cannot be undone.

Redo -- Does the action of the last Undo action again.

Cut -- Deletes the highlighted characters and puts them on the Clipboard.

Copy -- Copies the highlighted characters to the Clipboard.

Copy Qualified Name -- Copies the fully qualified class name of the custom helper superclass, if any, in the selected script, to the Clipboard.

Paste -- Inserts at the cursor any characters that were previously cut or copied to the Clipboard.

Delete -- Removes the highlighted characters.

Select All -- Highlights all the characters in the currently displayed script or file.

Expand Selection To -- Enables you to quickly select Java™ code in a syntax-aware way. For information, see the online *Java™ Development User Guide*.

Find/Replace -- Displays the Find/Replace dialog box, which enables you to search for a string of text in a script and substitute an alternate string.

Find Next -- Finds the next occurrence of the text selected in the Java™ Editor.

Find Previous -- Finds the previous occurrence of the text selected in the Java™ Editor.

Incremental Find Next -- Finds the next occurrence of the text you type. For information, see the online *Java™ Development User Guide*.

Incremental Find Previous -- Finds the previous occurrence of the text you type. For information, see the online *Java™ Development User Guide*.

Add Bookmark -- Displays the Add Bookmark dialog box, which enables you to insert a marker that points to a specific place in a script or file.

Add Task -- Displays the New Task dialog box, which enables you to associate a task with a specific location in a resource. For information, see the online *Java™ Development User Guide*.

Smart Insert Mode -- Toggles Smart Insert Mode on and off.

Expand Selection To -- Enables you to quickly select Java™ code in a syntax-aware way. For information, see the online *Java™ Development User Guide*.

Show Tooltip Description -- Displays the value of a hover that would appear at the current cursor location. For information, see the online *Java™ Development User Guide*.

Content Assist -- Assists you when writing Java™ code or Javadoc comments. For information, see the online *Java™ Development User Guide*.

Quick Fix -- Displays solutions for certain problems underlined with a problem highlight line. For information, see the online *Java™ Development User Guide*.

Word Completion -- Enables you to auto-complete names of elements in your script syntax. For information, see the online *Java™ Development User Guide*.

Parameter Hints -- Displays parameter type information. For information, see the online *Java™ Development User Guide*.

Set Encoding -- Changes the encoding of the currently shown text content to Default (inherited from container: CP1252), CP1252, ASCII, Latin 1, UTF-8, UTF-16 (big-endian), UTF-16 (little-endian), UTF-16, or Others. For information, see the online *Java™ Development User Guide*.

Source menu

The **Source** menu contains Eclipse commands that are not applicable to Rational® Functional TesterHCL OneTest™ UI.

Refactor menu

The **Refactor** menu contains Eclipse commands that are not applicable to Rational® Functional TesterHCL OneTest™ UI.

Navigate menu

The **Navigate** menu contains Eclipse commands that are not applicable to Rational® Functional TesterHCL OneTest™ UI.

Search menu

The **Search** menu contains Eclipse commands that are not applicable to Rational® Functional TesterHCL OneTest™ UI.

Project menu

The **Project** menu contains Eclipse commands that are not applicable to Rational® Functional TesterHCL OneTest™ UI.

Script menu

The **Script** menu has the following options:

Run -- Plays back the Functional Test script currently displayed in the Java™ Editor. For information see Running a Script from Functional Test on page 1445.

Debug -- Launches the current script and displays the Test Debug Perspective, which provides information as the script debugs. For information, see Debugging Scripts on page 1449.

Add Script Using Recorder -- Record a Functional Test script on page 877.

Insert Recording -- Starts recording at the cursor location in the current script on page 881, which enables you to start applications, insert verification points, and add script support functions.

Open Test Object Map -- Displays the test object map associated with the script currently displayed in the Java™ Editor.

Update Script Helper -- Updates the script's helper file (*ScriptHelper.java) to reflect changes made to the template for the selected test asset.

Insert Verification Point -- Displays the Select an Object page of the Verification Point and Action Wizard, which enables you to select an object in your application you want to perform a test on.

Insert Test Object -- Displays the Select an Object dialog box, which enables you to select test objects to add to the test object map and a script.

Insert Data Driven Commands -- Displays the Data Drive Actions page of the dataset Population wizard, which enables you to select the objects in an application-under-test to data-drive an application on page 938.

Find Literals and Replace with dataset Reference -- Replaces literal values with a dataset reference in a test script to add realistic data to an existing test script on page 950.

Configure menu

The **Configure** menu has the following options:

Configure Applications for Testing -- Displays the Application Configuration Tool on page 1735, which enables you to add and edit configuration information on page 796 -- such as name, path, and other information that Rational®

Functional TesterHCL OneTest™ UI uses to start and run the application -- for the Java™ and HTML applications you want to test.

Enable Environments for Testing -- Displays the Enable Java™ Environments (JRE) / Web Browsers / Eclipse Platforms for Testing dialog box, which you use to enable Java environments on page 778 and browsers on page 780 and to configure your JRE on page 797s and browsers on page 798.

Configure Object Recognition Properties -- Displays the Object Properties Configuration Tool, which enables you to Object Properties Configuration Tool on page 1883.

Configure Unexpected Windows -- Displays the Configuring how to handle unexpected windows during playback on page 1438, which enables you to configure how unexpected windows that open during script playback can be handled, to ensure smooth playback.

Run menu

The **Run** menu contains Eclipse menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Test Object Inspector menu item.

Test Object Inspector -- Displays test object information, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information.

Window menu

The **Window** menu contains Eclipse menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Open Perspective and Preferences menu items.

Open Perspective -- Open a another perspective in the Rational® Functional TesterHCL OneTest™ UI Debug window. For information, see the online *Workbench User Guide*.

Preferences -- Displays the Rational® Functional TesterHCL OneTest™ UI Preferences page, which enables you to customize Functional Test by setting preferences on page 822 for Rational® Functional TesterHCL OneTest™ UI on page 828, Highlight on page 831, Logging on page 834, Operating System on page 836, Playback on page 836, Playback Delays on page 839, Playback Monitor on page 840, ScriptAssure(TM)-Standard on page 841, Script Assure(TM)- Advanced on page 840, Recorder Preferences on page 843, Recorder Monitor Preferences on page 844, Workbench Preferences for Rational® Functional TesterHCL OneTest™ UI, on page 857 and Workbench Advanced Preferences for Rational® Functional TesterHCL OneTest™ UI on page 858.

Help menu

The **Help** menu has the following options:

Welcome -- Displays the Welcome to Rational® Software Delivery Platform HCL Products page, which contains information that will help familiarize you with Rational® Functional TesterHCL OneTest™ UI.

Help Contents -- Displays the online Rational® Functional TesterHCL OneTest™ UI table of contents and information, which you can use to navigate to various topics.

Search -- Displays the Search page, which enables you to search the *Product help*.

Dynamic Help -- Displays Java specific text editing support information.

Index -- Displays the Rational® Functional TesterHCL OneTest™ UI index.

Key Assist -- Display a list of keyboard shortcuts that you can use while working with Rational® Functional TesterHCL OneTest™ UI.

Tips and Tricks -- Display tips and tricks for Eclipse Java™ Development tools, Eclipse platform, and Eclipse plug-in development environment.

Functional Test Help -- Displays the online Rational® Functional TesterHCL OneTest™ UI table of contents and information.

Functional Test Proxy SDK -- Displays the Rational® Functional TesterHCL OneTest™ UI proxy software development kit (SDK) help.

Functional Test API Reference -- Displays an overview of the online *Functional Test API Reference Guide*, which you can use to navigate to various topics.

Documentation Feedback -- Displays the Documentation Feedback page, where you can provide feedback about the Rational® Functional TesterHCL OneTest™ UI documentation.

Web Resources -- Displays troubleshooting and support information that will enable you to fix problems and find additional resources.

Cheat Sheets -- Displays the cheat sheets for using Rational® Functional TesterHCL OneTest™ UI.

Process Advisor -- Displays best practices and guidance that will help you learn how to create projects and test scripts using Rational® Functional TesterHCL OneTest™ UI.

Process Browser -- Displays best practices topics and tasks from the Process Advisor.

Local Help Updater -- Displays the Local Help Updater which will help you download and update help content from the Help updater site.

Manage Licenses -- Displays the Manage Licenses dialog box which will help you to apply licenses, or update your license status.

Check for Updates and **Install New Software**-- Enable you to update products and to download and install new features. Manage the configuration of Rational® Functional TesterHCL OneTest™ UI. Scan for updates for all installed features, add an extension location, view installation history, and show activities that caused the creation of the configuration. For information, see the online *Workbench User Guide*.

IBM Installation Manager -- Displays IBM Installation Manager, which you can use to install, update, modify, roll back or uninstall software, or manage licenses for Rational software.

Rational Functional TesterHCL OneTest UI -- Displays information about the current version of the Rational® Functional TesterHCL OneTest™ UI you are running.

Rational® Functional TesterHCL OneTest™ UI General page

You use the Rational® Functional TesterHCL OneTest™ UI General page to set all product time options. These options are useful to accommodate different computer speeds.

The General page has the following controls:

Limit Record/Playback to StartApp applications only: Select this option to limit the recording and playing back to StartApp application only.

Automatic enablement: Automatic enablement is activated by default. Deselect the checkbox if you want to statically enable each test environment. This is useful for improving the performance of tests.

Multiply all time options by: Enter any real number by which you want to multiply all Rational® Functional TesterHCL OneTest™ UI preferences or options that take an amount of time as an argument. For example, enter .5 to make all Rational® Functional TesterHCL OneTest™ UI time options twice as fast. This option affects all the following controls:

General Playback

- **Maximum time to attempt to find Test Object**
- **Pause between attempts to find Test Object**
- **Timeout used in waitForExistence( ) method**
- **Retry timeout used in waitForExistence( ) loop**

Delays

- **Delay before mouse up**
- **Delay before mouse move**
- **Delay before mouse down**
- **Delay before key up**
- **Delay when hover**
- **Delay after top level window activates**
- **Delay before key down**
- **Delay before performing Test Object action**

General Recorder

- **Delay before recording a mouse action**

Delay before recording a keystroke

Use Default

Clear this check box to edit the value in the **Multiply all time options by** field. Select this check box to restore the default value.

Restore Defaults

Restore the default values on this page.

Apply

Save your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, click **Functional Test**.

To open: Click **Tools > Options**. In the left pane, click **IBM Rational Rational Functional Tester**.

Related information

Restricting the actions during recording and playback

Rational® Functional TesterHCL OneTest™ UI Project Properties page

Use this page to change your default script helper superclass.

Rational® Functional TesterHCL OneTest™ UI uses the helper superclass for all the scripts you create in this project. You can change the superclass for an individual script by specifying it in the Script Properties page on page 1773.

By default, all Rational® Functional TesterHCL OneTest™ UI scripts extend the RationalTestScript class and inherit a number of methods (such as callScript). You can create your own helper superclass to add methods or override the methods from RationalTestScript. Use this properties page to change the default helper superclass for a project.

Default Script Helper superclass -- Enter the fully qualified class name of your custom helper superclass in this field. Note that your helper superclass must extend RationalTestScript.

If you change your superclass and want to reset it to RationalTestScript, you can either type RationalTestScript in the superclass field or clear the field. Leaving this field blank resets the script to use RationalTestScript.

To open: In the Rational® Functional TesterHCL OneTest™ UI Projects view, select a project, right-click, and click **Properties**. Click **Rational Functional TesterHCL OneTest UI Project**.

Rational® Functional TesterHCL OneTest™ UI Projects view

The Functional Test Projects view, which is the left pane of the Functional Test Perspective, lists test assets for each project.

The following icons appear in the Projects view pane:

 Folders

 Simplified test scripts

 Java test scripts

 Shared test object maps

 Log folders

 Logs

 Java™ file

The Functional Test Projects view banner has the following buttons:

The **Connect to a Functional Test Project**  button allows you to connect to an existing Functional Test project.

The **Refresh Projects** button  enables you to repaint the display to reflect changes.

The **Synchronize with Editor** button  scrolls in the tree hierarchy to the name of the script currently displayed in the Java™ Editor.

Double-clicking a script in the Projects view opens the script in the Java™ Editor.



Note: If there are no projects in the Projects view, instructions display informing you how to create a new Functional Test project or connect to an existing Functional Test project. If you do not select any item in the Projects view, and right-click in the Projects view, a menu is displayed, from which you can create a new Functional Test project, connect to an existing Functional Test project, or refresh the project(s).

Right-clicking on a project or test asset listed in the Projects view displays various menu options, which are listed here in alphabetical order:

Add Empty Script -- Displays the Create an empty Functional Test script dialog box, which enables you to create a script on page 879 you can use to manually add Java™ code.

Add Script Using Recorder -- Displays the Record a Functional Test script dialog box, which enables you to enter information about the new script and start recording on page 877.

Add Test Folder -- Displays the Create a Test Folder dialog box, which enables you to create a new Functional Test folder on page 862 for the project or under an existing folder.

Add Test Object Map -- Displays the Create a Test Object Map dialog box, which enables you to add a new test object map to a project.

Add Test dataset -- Displays the Create a Test dataset dialog box, which enables you to create a new test dataset. on page 943

Clear As Project Default -- Removes the default designation from the selected test object map. To set the default designation, right-click the test object map in the Rational® Functional TesterHCL OneTest™ UI Projects view and select **Set As Project Default** .

Debug -- Launches the current script and displays the Test Debug Perspective, which provides information as the script debugs.

Delete -- Enables you to delete the selected test asset..

Disconnect Project -- Disconnects a Functional Test project on page 859, which removes it from the Functional Test Projects view.

Export -- Enables you to export project items for the selected log, project, or script.

Failed Verification Points -- Opens the selected verification point actual results file in the Verification Point Comparator on page 923, where you can compare and edit the data. See About Logs on page 1497.

Final Screen Snapshot -- Available when the log of a script that failed on its last run is selected. Opens the screen snapshot image taken at playback failure. See Screen snapshot on playback failure on page 1450.

Import -- Enables you to import project items for the selected log, project, or script.

Insert as "callScript" -- Available when a script is selected, inserts the `callScript ("scriptname")` code in the current script at the cursor location. See Calling a Script from a Functional Test Script on page 891.

Insert contained scripts as "callScript" -- Available when a project is selected, displays a message that enables you to choose **Yes** or **No** . **Yes** inserts a callScript command for all scripts in the project, including the selected folder(s) and all subfolders. **No** inserts a callScript command only for scripts in the selected folder(s). See Calling a Script from a Functional Test Script on page 891.

Merge Objects into -- Displays the Merge Test Objects into the Test Object Map page, which enables you to merge multiple test object maps.

Open -- Opens the selected script or Java™ class in the Java™ Editor or opens the selected test object map.

Open Log -- Opens the selected log. See About Logs on page 1497.

Open Test Object Map -- Enables you to display the selected test object map.

Properties -- Displays information about the selected Functional Test project, test object map, test folder, script, or log.

Rename -- Displays the Rename dialog box.

Reset Java Build Path -- Synchronizes the .JAR files in the Customization folder (C:\Program Files\IBM\Rational \FunctionalTester\Customization) (C:\ProgramData\HCL\UI\Customization) with the Java™ build path for

Functional Test projects. The Java™ build path appears on the Java™ Build Path page of the Properties dialog box. For information, see the online *Java™ Development User Guide*.

Run -- Plays back a selected Functional Test script on page 1445.

Set as Project Default -- Indicates the selected test object map as the default in a variety of wizards and dialog boxes, such as the Select Script Assets on page 1834 page of the Record New Functional Test Script and the Create Empty Functional Test Script wizards, and the Copy Test Objects to New Test Object Map on page 1745 page of the Create new Test Object Map wizard. To remove the designation, right-click the test object map in the Projects view and select **Clear As Project Default** .

Show in Navigator -- Reveals the currently selected element's underlying resource (or the current editor's underlying resource) in the Navigator view. For information, see the online *Java™ Development User Guide*.

Team -- Enables you to add test elements to source control, check out elements, check in elements, undo a checkout, get latest version, show checkouts, display the history of an element, share a project, or compare versions or elements.

To open: Rational® Functional TesterHCL OneTest™ UI automatically displays the Projects view (by default) in the Functional Test Perspective.

Rational® Functional TesterHCL OneTest™ UI Script Properties page

Use this page to change your script helper superclass, test dataset, or dataset iteration for an individual script.

Rational® Functional TesterHCL OneTest™ UI uses a default helper superclass for all the scripts you create in a project. This superclass is listed in the Functional Test Project Properties Page on page 1770. You can use a different superclass for an individual script by changing it in this Script Properties page.

By default, all Functional Test scripts extend the RationalTestScript class and inherit a number of methods (such as callScript). You can create your own helper superclass to add methods or override the methods from RationalTestScript. Use this properties page to change the default helper superclass for the selected script.

Test Object Map -- Displays either **Private** to indicate that the script test object map is private or the name of the shared test object map.

Helper Superclass -- Enter the fully qualified class name of your custom helper superclass in this field. Note that your helper superclass must extend RationalTestScript.

If you change your superclass and want to reset it to RationalTestScript, you can either type RationalTestScript in the superclass field or clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

Test dataset -- Click **Browse** to change the dataset associated with a script.

dataset Record Selection Order -- Specifies how a test script accesses records in its associated dataset when you play back the test script. Click the **dataset Record Selection Order** arrow to change the test dataset iterator.

Types of dataset iterators:

- **Sequential** -- Makes a test script access records in the dataset in the order that they appear in the dataset. This is the default dataset iterator.
- **Random** -- Makes a test script access records in the dataset randomly. A random iterator accesses every record in the dataset once.

To open: In the Projects View, select a script, right-click, and click **Properties**. Click **Functional Test Script**.

Rational® Functional TesterHCL OneTest™ UI Script Templates Properties page

You can use the Functional Test Script Templates Properties Page to edit a Functional Test script template.

Select template type

Lists all the types of templates that you can edit.

Script: Header of the file

Use to customize the layout of new script files.

Script: Comment for Test Object

Use to customize a test object comment line inserted into a script by the recorder.

Script: Comment for top level Test Object

Use to customize a top-level test object comment line inserted into a script by the recorder.

Script: HTML Test Object Name

Use to customize the names of HTML test objects in a script.

Script: Java Test Object Name

Use to customize the names of Java™ test objects in a script.

Script: .Net Test Object Name

In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, use to customize the names of .NET test objects in a script.

Script: Windows Test Object Name

In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, use to customize the names of Windows® test objects in a script.

VP: Verification Point Default Name

Use to customize the names of verification points Rational® Functional TesterHCL OneTest™ UI generates by default in the Verification Point and Action Wizard.

Script Helper: Header of the file

Use to customize the layout of a helper class when auto-generated.

Script Helper: Test Object Method

Use to customize the layout of test object methods in the helper class.

Script Helper: Verification Point Method

Use to customize the layout of verification point methods in the helper clas

Script Helper Superclass (Rational® Functional TesterHCL OneTest™ UI, Java™ Scripting)

Use to customize the layout of the script helper superclass.

Script Helper Base Class (Rational® Functional TesterHCL OneTest™ UI, VB.NET Scripting)

Use to customize the layout of the script helper base class.

Open current template in Editor

Click to use the appropriate editor to customize the script template.

Restore Defaults

Restores all script templates to the original defaults of the database when you created it. All your edits are lost.

Apply

Applies your edits to the template on which you are working. If you are making extensive edits, it is a good idea to apply your changes to save your edits as you make them or to each template as you complete changes.

To open in Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration: From the Projects view, right-click a Functional Test project, click **Properties**, and click **Functional Test Script Templates**.

To open inRational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration: From the Solution Explorer, right-click a Functional Test project, click **Properties**, and click **Functional Test Script Templates**.

Rational® Functional TesterHCL OneTest™ UI Show Checkouts View

You can use the Functional Test Show Checkouts View to list checked-out elements in one or more projects.

By default, the Functional Test Show Checkouts view lists the elements you have checked out in the current directory and in the current view.

Note: The status bar in the Rational® Functional TesterHCL OneTest™ UI Perspective displays the project or projects you selected to search for checked-out elements.

Type

Displays the file type of each checked-out element. The icons for the file types are:

Icon	Description
	Java™ class files

Icon	Description
	Object map
	Test script
	Functional Test script template files

Element Name

Displays the name of the checked-out element. If an element is in a folder, the folder appears as *<foldername>.<filename>*. For example, `Myfolder.myfile`.

`Rational Functional Tester ProjectHCL OneTest UI Project`

Displays the name of the project that contains a checked-out element. Click **X** to close the **Show Checkouts View**.

To open: In the Projects view, right-click one or more projects, and then click **Team > Show Checkouts**.

Rational® Functional TesterHCL OneTest™ UI Show History View

You can use the Rational® Functional TesterHCL OneTest™ UI Show History View to display the history of an element under source control.

Type

Displays the file type of the element. The icons for the file types are:

Icon	Description
	Java™ class files
	Object map
	Test script
	Functional Test script template files

Name

Displays the name of the object.

Date

Displays the date and time of the revision.

Comment

Displays the first few characters of the comment. To see the entire comment, right-click an entry and click **Show Comment**.

User Name

Displays the name of the user who made the change.

Event

Displays the nature of the change.

To open: In the Projects view, right-click the test asset, and then click **Team> Show History**.

The Rational® Functional TesterHCL OneTest™ UI toolbar

This topic describes all the buttons on the Rational® Functional TesterHCL OneTest™ UI toolbar.

The Rational® Functional TesterHCL OneTest™ UI toolbar has the following buttons:

 **New** -- Displays the appropriate dialog box that enables you to create a new project, Functional Test project on page 858, to record a Functional Test script on page 877, Functional Test empty script on page 879, to create a test folder on page 862, test dataset on page 943, test object map, or helper superclass on page 889. Click to display the New dialog box or click to display the list of items to create.

 **Save** -- Saves any changes you made to the script currently displayed in the Java™ Editor. For information, see Saving Functional Test Scripts and Files. on page 898

 **Print** -- Displays the Print dialog box, which enables you to print the current script.

 **Create a Functional Test Project** -- Displays the Create a Functional Test project dialog box, which enables you to generate a new project on page 858.

 **Connect to an existing Functional Test Project** -- Displays the Connect to a Functional Test project dialog box, which enables you to specify the project you want to use and its location. You must first connect to an existing Functional Test project on page 859 before you can use it.

 **Create an Empty Functional Test Script** -- Displays the Create an empty Functional Test script dialog box, which enables you to create a script on page 879 you can use to add Java™ code.

 **Create a Test Object Map** -- Displays the Create a Test Object Map dialog box, which enables you to Creating a new test object map to a project.

 **Create a Test dataset** -- Displays the Create New Test dataset dialog box, which enables you to create a new test dataset on page 943.

 **Create a Test Folder** -- Displays the Create a New Functional Test Test Folder dialog box, which enables you to create a new folder on page 1745 either for the project or for an existing folder.

 **Record a Functional Test Script** -- Displays the Record a Functional Test script dialog box, which enables you to enter information about the new script and start recording on page 877.

 **Insert Recording into Active Functional Test Script** -- Starts recording at the cursor location in the current script on page 881, which enables you to start applications, insert verification points, and add script support functions.

 **Run Functional Test Script** -- Plays back the Functional Test script currently displayed in the Java™ Editor. Click to begin program execution at method Main in the current script or click ▼ to display the list of scripts to run. *Note:* Since method Main is not in Functional Test user scripts, you will receive an error if you select **Run > Java Application**. For information, see Running Scripts on page 1445.

 **Debug Functional Test Script** -- Launches the current script and displays the Debug Perspective, which provides information as the script debugs. Click to begin debugging at method Main in the current script or click ▼ to display the list of scripts to debug. *Note:* Since method Main is not in Functional Test user scripts, you will receive an error if you select **Debug > Java Application**. For information, see Debugging Scripts on page 1449.

 **Configure Applications for Testing** -- Displays the Application Configuration Tool on page 1735, which enables you to add and edit configuration information -- such as name, path, and other information that the product uses to start and run the application -- for the Java™ and HTML applications you want to test.

 **Enable Environments for Testing** -- Displays the Enable Java™ Environments (JRE)/Web Browsers/Eclipse Platforms for Testing dialog box, which you use to on page 796 enable Java environments on page 778 and on page 796 browsers on page 780 and on page 796 to configure your JRE on page 797s and browsers on page 798.

 **Open Test Object Inspector** -- Displays the Test Object Inspector tool, which enables you to display test object information on page 885, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information.

 **Insert Verification Point into Active Functional Test Script** -- Displays the **Select an Object** page of the Verification Point and Action wizard, which enables you to select an object in your application you want to perform a test on.

 **Insert Test Object into Active Functional Test Script** -- Displays the Select an Object dialog box, which enables you to select test objects to Adding a test object to an object map and a script.

 **Insert Data Driven Commands into Active Functional Test Script** -- Displays the Data Drive Actions page of the dataset Population Wizard, which enables you to select the objects in an application-under-test to data-drive an application on page 938.

 **Find Literals and Replace with dataset Reference** -- Replaces literal values with a dataset reference on page 950 in a test script, which enables you to add realistic data to an existing test script.

 **External Tools** -- Enables you to run or configure an external tool that is not part of Workbench. Click to create, manage, and run configurations or click ▼ to display the list of external tools to run or configure. For information, see the online *Workbench User Guide*.

 **Search** -- Displays the Search dialog box, which enables you to search for text or Java™ code.

Last Edit Location -- Navigates to the previous location where you edited the script.

Back -- Navigates to the previous file you viewed. Click to navigate to the previous file you viewed or click to display the list of files to navigate backward.

Forward -- Navigates to the next file you viewed. Click to navigate to the next file you viewed or click to display the list of files to navigate forward.

Highlight page

You use the Highlight page to specify how you want Rational® Functional TesterHCL OneTest™ UI to emphasize test objects in applications-under-test when you select them in a test object map or in the Script Explorer. These settings also control how Rational® Functional TesterHCL OneTest™ UI highlights objects you select with the Verification Point and Action Wizard and the Insert a GUI Object into the Object Map dialog box.

You can also change these settings in the test object map by clicking **Preferences > Highlight** on the test object map menu.

The Highlight page has the following controls:

Color

Click to display a color selection palette from which you can select a color to use to indicate selected test objects. The button displays the color currently in use.

Border Width (in pixels)

Move the slider from **Thin** to **Wide** to set the width of the border around the selected object.

Flash Speed

Move the slider from **Slow** to **Fast** to set the rate at which the border around a selected object flashes when selected.

Display Time

Move the slider from **Short** to **Long** to set the length of time to highlight the border.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, expand **Functional Test** and click **Highlight**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Highlight**.

Import dataset dialog box

Use the Import dataset dialog box to create a dataset from an existing Rational® Functional TesterHCL OneTest™ UI dataset or a .csv file.

Import From

Type the path and file name of an existing Rational® Functional TesterHCL OneTest™ UI dataset, or a .csv file, or click the **Import From** arrow to display a list of ten recently imported files, and then click an item from the list.

Browse

Click to select the path and filename of an existing Rational® Functional TesterHCL OneTest™ UI dataset to import. When you browse for a dataset, click the **Files of type** arrow and select either a .csv file type, a .rftdp file type for a public dataset, or a /rftxdp file type for a private dataset.

.CSV Format options

Use these options to import a .csv file.

Field Separator

Type the field separator character to use in the file you want to import. The field separator character that appears must be the same as the one used in the .csv file you are importing as a dataset. If you are not sure which field separator character to specify, use a text editor such as Notepad to open the .csv file and see the field separator character that is used.

First Record is Variable Information

Select to make the first row of the imported data, the column heading of a new dataset. If **First Record is Variable Information** is unchecked, the first row in the .csv file is imported as data and the headers use the default headers, such as data0, data1. The **First Record is Variable Information** setting is unavailable if the file you select to import is not a valid .csv file.

Note: When you import a .csv file that was initially a dataset, you must select **First Record is Variable Information** to ensure that the data imports correctly.

To open: Click **Create a Test dataset** (). For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, click **Next**. For Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, click **Open**.

Note: The **Drag Hand Selection** method is not available on Linux environments such as Ubuntu and Red Hat Enterprise Linux (RHEL). You must use the **Test Object Browser** method on Linux environments.

Import Project Items page

You use the Import Project Items page to import project items such as scripts, test object maps, Java™ files or Visual Basic files, and datasets into a Functional Test project.

The Import Project Items page has the following controls:

Transfer file

Enter or navigate to the data transfer file name .rftjdtr .rftvdtr that was used to export the project items. To view and work with items in the data transfer file, you can use any file compression program that supports the .zip format. You do not have to extract files in the .rftjdtr .rftvdtrfile before importing.

Select the import location

Lists all Functional Test projects. Use the navigation buttons (Home  , Back  , and Go Into ) to select the appropriate path to the project into which you want to import project items.

Back

Returns to the Import wizard.

Finish

Adds all the project items from the data transfer file into the project you specified. If the project already contains any of the assets you are importing, Rational® Functional TesterHCL OneTest™ UI displays the Select items to overwrite page. Select the items to overwrite in the project or clear the items that you do not want to overwrite.

To open: Right-click the project in the Projects view, and click **Import**. In the Import wizard, select **Functional Test Project Items** and click **Next**. Right-click the project in the Solution Explorer and click **Import**.

Insert Data Drive Actions dialog box

You can use the Insert Data Drive Actions dialog box to select the test objects and actions you want to data drive.

 **Object Finder Tool**

Use to select an object and all the descendents of the selected object. Using the Object Finder tool is the most common and direct method of selecting an object.

 **Selection Wizard**

Click to use the Object Finder Tool method and its options, or the Object Browser method.

Data Driven Commands

Displays information about the test object or objects that you selected to data drive the testing of your application-under-test. You can place your pointer over a row in this table to view the line of code that Functional Test inserts into the test script to data drive the application-under-test.

Role

Displays an icon that represents the type of test object you select to data drive an application-under-test.

Test Object

Displays the name of a test object.

Command

Displays the command that you can perform on a particular test object.

Variable

Displays the name of the heading for the variable as the name appears in a dataset. Type a descriptive name for the variable or select an existing variable name by clicking the drop-down arrow. Descriptive headings make it easier to add data to the dataset.

Initial Value

Displays the initial value of a test object. Double-click to change the initial value of a test object to test your application with different values.

The following icons appear to the right of the Data Driven Commands table:

 Click to move the selected row earlier in the order of execution in the Data Driven Commands table.

 Click to move the selected row later in the order of execution in the Data Driven Commands table.

 Click to delete a selected row from the Data Driven Commands table.

 Click to highlight a test object in the application-under-test. Select a test object in the Data Driven Commands table, and then click this icon.

 Click to display or hide the recognition and administrative properties for a selected test object.

Selected Command Description

Displays recognition and administrative properties about the test object selected in the **Data Driven Commands** area of the Insert Data Drive Actions dialog box. Appears when you click  .

Recognition

Lists the property, value, and weight of a selected test object. You can use this information to confirm that you have selected the correct test object to data drive.

Administrative

Displays internal administrative data of a selected object. You cannot edit this data. Use this data to locate and manage the test object in the context of the associated script. You can use the administrative information to determine what test object this is in the associated application-under-test.

To open: Click **Insert Data Drive Commands** () on the Recording toolbar.

Insert getProperty Command page

Use to get a single property value for the selected object. Functional Test puts a getProperty command into your script and returns the value during playback. This information is useful when you need to make a decision based on the property. For example, you might want to query whether a button was enabled.

When you select an object, the property list is built and displayed in the **Property Name** and **Value** fields. Select the property that you want to get. Click the **Next** button to proceed. The getProperty command is written into your script at the point you inserted it.

Note: If your object has no properties, this page is disabled.

Insert Properties Verification Point Command page

Use to create a Properties verification point for the selected object. The Properties verification point tests the properties in your object when you play back your script. The object name is listed on the page. This verification point tests all properties of the object. You can edit the properties list later if you want to test only some of the properties.

Use the **Include Children** field to specify whether to include the properties of any child objects. **None** tests the object only (no children), **Immediate** tests the object and any immediate children (one level down), and **All** tests the object plus all of its children down the entire hierarchy.

Under **Verification Point Name**, accept the suggested default, or type a new name in the box.

Use the **Use standard properties** option to specify whether to use standard property types. If you are testing Java™, all properties are standard. Clear the option only if you are testing HTML and want to test browser-specific properties.

Use the **Include Retry Parameters** to set a retry time for a verification point during playback to check for its existence. The retry option is useful when playback does not find the verification point in your application. To set a retry time, either use the default or set your own time. **Maximum Retry Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI tries again for the verification point to be shown in your application during playback. **Retry Interval** is the number of seconds between times that Rational® Functional TesterHCL OneTest™ UI will check for the verification point during the wait period.

When you select **Include Retry Parameters**, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the verification point in your application every 2 seconds, for up to 20 seconds. To set your own time, clear the default fields and provide your own values for **Maximum Retry Time** and **Retry Interval**. When you click **Finish**, the retry for verification point is written into your script and occurs on future playbacks.

Note: When you insert a Properties verification point without recording, the Include Retry Parameters option does not appear on the Insert Properties Verification Point Command page.

To proceed with the verification point, click **Next**. For more information, see Creating a Properties Verification Point on page 904.

Note: If your object has no properties, this page is disabled.

Insert Verification Point Data Command page

Use to create a Data verification point for the selected object. The Data verification point tests the data in your object when you play back your script. The object name is listed at the top of the page. The list of tests shown in the **Data Value** field depends on information provided by the object proxy. Select the data value that you want to test.

Under **Verification Point Name**, accept the suggested default, or type a new name in the box.

Use the **Include Retry Parameters** to set a retry time for a verification point during playback to check for its existence. The retry option is useful when playback does not find the verification point in your application. To set a retry time, either use the default or set your own time. **Maximum Retry Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI tries again for the verification point to be visible in your application during playback. **Retry Interval** is the number of seconds that Rational® Functional TesterHCL OneTest™ UI checks for the verification point during the wait period.

When you select **Include Retry Parameters**, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the verification point in your application every 2 seconds, for up to 20 seconds. To set your own time, clear the default fields and type in your own values for **Maximum Retry Time** and **Retry Interval**. When you click **Finish**, the retry for verification point is written into your script, and occurs on future playbacks.

Note: When you insert a Data verification point without recording, the Include Retry Parameters option does not appear on the Insert Verification Point Data Command page.

To proceed with the verification point, click **Next**. For more information, see Creating a Data Verification Point on page 907.

Note: If your object has no data, this page is disabled.

Insert waitForExistence Command page

Use to set a wait state for an object during playback to check for its existence. The waitForExistence command is useful when waiting for an object right after your application starts, or after other actions that may take a long time.

The selected object is listed at the top of the page. To set a wait state for it, either use the default or set your own time. **Maximum Wait Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI waits for the object to appear in your application during playback. **Check Interval** is the number of seconds between times that Rational® Functional TesterHCL OneTest™ UI checks for the object during the wait period.

When **Use the Defaults** is selected, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the object in your application every 2 seconds, for up to 120 seconds. To set your own time, clear the default field and provide your own values for **Maximum Wait Time** and **Check Interval**. When you click **Finish**, the wait-for object is

written into your script, and will occur on future playbacks. For more information, see Setting a Wait State for an Object on page 886.

Java editor

You use the Java™ Editor (the script window) to edit Java™ code.

The name of the script or class you are currently editing appears in a tab on the Java™ editor frame. An asterisk on the left side of the tab indicates that there are unsaved changes.

You can open several files in the Java™ editor and move between them by clicking on the appropriate tab.

If there is a problem with the code, a problem marker is displayed near the affected line.

Right-clicking in the Java™ Editor displays various menu options for working with scripts.

Refer to the online *Java™ Development Guide* for more information.

The Java™ Editor is automatically displayed (by default) in the Functional Test Perspective.

Java Environments tab of the Enable Environments dialog box

This dialog is opened by clicking **Configure > Enable Environments for Testing** from Rational® Functional TesterHCL OneTest™ UI. The **Java Environments** tab is used to enable Java™ environments and to add and configure Java™ environments. Information about enabling JRE's is presented first. Information about adding and configuring JRE's is presented below that.

For enabling Java environments:

The Java™ enabler is the **Java Environments** tab of the Enable Environments dialog box. The Java™ enabler must be run before you can use Rational® Functional TesterHCL OneTest™ UI to test Java™ applications. It modifies your Java™ environments to allow Java™ programs run under them to be tested by Rational® Functional TesterHCL OneTest™ UI. The enabler scans your hard disk drive(s) looking for Java™ environments. It allows Rational® Functional TesterHCL OneTest™ UI to "see" your Java™ environments by adding files into the directory where your Java™ Runtime Environments (JREs) are located.

The first time you run Rational® Functional TesterHCL OneTest™ UI, it automatically enables your Java™ environments. If you install a new Java™ environment, such as a new release of a browser or a JDK, you must rerun the enabler after you complete the installation of the Java™ environment. You can run the enabler any time from Rational® Functional TesterHCL OneTest™ UI by clicking **Configure > Enable Environments for Testing**. Note that the first time you run Rational® Functional TesterHCL OneTest™ UI it automatically enables the JVM of your browser's Java™ plug-in so that HTML recording works properly. If you install a different JVM, you must rerun the enabler to enable it.

Java Environments List

Rational® Functional TesterHCL OneTest™ UI is shipped with a JRE that is automatically enabled during your installation. It is called "Default JRE," and will appear in this list. Other than the Default JRE, this list is populated by the **Search** or **Add** buttons. It displays the list of Java™ environments that the enabler locates on your hard disk drive(s). After the name, the enabler indicates in parentheses whether that environment is currently enabled.

Select All Button

Use this to select all the JREs that are listed in the **Java Environments** list. This is useful if you want to enable or disable all the environments. To clear them all, click any of the individual environments.

Search Button

Click this button to have Rational® Functional TesterHCL OneTest™ UI search your hard disk drive(s) for Java™ environments. This opens the Search for Java™ Environments dialog box. Choose one of the search options in that dialog and click the **Search** button. **Note:** You should not use the **Search All Drives** option to find JREs on Linux® or UNIX® systems. Instead use the **Search In** option and browse for the JRE. See Enabling Java Environments topic for information on the search options. When the search is complete, the **Java Environments** list is populated with all found environments. At least the first time that you use Rational® Functional TesterHCL OneTest™ UI, use the **Search** button to locate all JREs on your system. After the initial search, it will list any JREs that were already enabled, plus any new ones it finds.

Add Button

Click this button to locate Java™ environments individually by browsing. It brings up the Add Java™ Environment dialog box to locate a JRE on your system. To select a JRE, you can point to the JRE's root directory or any file under the root directory. The JRE you select will be added to the **Java Environments** list. You can also use the **Search In** option in the Search for Java Environments dialog box to browse for a JRE.

Remove Button

To remove an environment from the **JREs** list, select it, then click **Remove**.

Set as Default Button

Use this to choose which JRE you want to be your default environment used in playback. Select the JRE in the list, and click the button. That JRE will then become the default, and will be indicated in parentheses after the name. You can change the default any time by coming back to this tab.

Enable Button

Use this button to enable selected Java™ environments for testing with Rational® Functional TesterHCL OneTest™ UI. Select the JRE(s) to enable in the list, then click **Enable**. The modifications to the JRE(s) are done at this time. This includes adding Rational® Functional TesterHCL OneTest™ UI classes to the JRE and modifying the accessibility.properties file, which tells Java™ to load Rational® Functional TesterHCL OneTest™ UI classes when it runs a Java™ application. Once a JRE is enabled, that will be indicated in parentheses after its name in the list.

Disable Button

Use this button to disable selected Java™ environments for testing with Rational® Functional TesterHCL OneTest™ UI. Select the JRE(s) to disable in the list, then click **Disable**. This undoes all the modifications made by **Enable**. Once disabled, that will be indicated in parentheses after each JRE name in the list.

Test Button

You can test that your JRE is enabled properly by clicking the **Test** button in the enabler. This opens the JRE Tester. It will report the JRE version, JRE vendor, and whether it is enabled successfully.

Note: If your JRE is not enabled, you will be able to tell because the Record Monitor is blank when you try to record against a Java™ application. You should leave the Record Monitor in view while recording for this reason. If you see this symptom, you need to run the enabler.

Note: To enable browsers for HTML testing, click the **Web Browsers** tab of the enabler and click the **Help** button, or see Enabling Web Browsers topic.

For adding and configuring Java environments:

The **Java Environments** tab is also used to add and edit JRE configurations. To edit the information on an existing JRE, click the name of the JRE in the **Java Environments** list. To add a new JRE, click the **Search** or **Add** button. Use the **Set as Default** button to set one of the JREs as your default to be used during playback. Whether editing or adding, make your changes, then click **OK** for the changes to be saved.

Java Environments List

Select the JRE that you want to edit or view. Its information will then appear to the right of the list. The information fields are described below. If your JRE is not in the list yet, click **Search** or **Add** to find and enter it.

The JRE that has default listed after it in parentheses in the list is the default JRE. It will be used in all Java™ testing unless you change this setting in the properties of a specific application.

Detailed Information for *Java Environment*

Contains the following fields:

Name -- This is the logical name of your JRE. For example, "Java2" may be used to represent JDK 1.2.2. This will default to the name at the end of your class path. You can edit this name.

Path -- This is the full path to the root of the JRE installation. If the path is incorrect, you will get an error message.

Run Command -- The command line that runs your Java™ applications with this JRE. The default is "java" for most JRE's. The JRE that comes with Rational® Functional TesterHCL OneTest™ UI is "javaw."

Run Options -- This is blank by default. If you need to use any special flags for this JRE, enter them here.

Search Button

Click **Search** to add all your JREs into the **Java environments** list. This opens the Search for Java™ Environments dialog box. Choose one of the search options in that dialog and click the **Search** button. **Note:** You should not use the **Search All Drives** option to find JREs on Linux® or UNIX® systems. Instead use the **Search In** option and browse for the JRE. Rational® Functional TesterHCL OneTest™ UI will enter all the detailed information on each JRE, except for the **Run Options** field.

Add Button

Click **Add** to browse for a new JRE to add to the list. The Add Java™ Environment dialog appears. Browse to the JRE you want to add. You can select any file under the root of the JRE, or the root directory itself. With the file selected, click the **Add** button. The JRE will then show up in the **Java Environments** list and you can edit its configuration information if necessary. Note: it is quicker to use the **Search** button and let Rational® Functional TesterHCL OneTest™ UI find and enter your environments if you are entering multiple JREs.

Set as Default Button

Use this to choose which JRE you want to be your default environment used in playback. Select the JRE in the list, and click the button. That JRE will then become the default, and will be indicated in parentheses after the name. You can change the default any time by coming back to this tab. You can also override the default environment for a specific application, by indicating it in the **JRE** field in the Application Configuration Tool.

Remove Button

To remove an environment from the **JREs** list, select it, then click **Remove**.

OK Button

You must click **OK** when you are finished to save the additions or edits you made on this tab.

For more information, see Configuring Java Environments for Testing topic.

Apply Button

If you want to apply edits you make in this dialog box before you exit the dialog, click **Apply** . If you click **Cancel**, any changes you made before you clicked **Apply** will be saved, and changes made after will be canceled.

Log Entry tab: Script Support Functions dialog box

You use the Log Entry tab to insert a log message into a Functional Test scripts and indicate whether it is a message, warning, or an error. During playback, Rational® Functional TesterHCL OneTest™ UI inserts this information into the log.

The **Log Entry** tab has the following controls:

Message to write to the log

Enter the text you want to include in the log.

Result

Select the type of message you want to add to the log. The result type will be displayed in the log.

Information

Indicates that the text will be entered as a message.

Warning

Indicates that the text will be entered as a warning. The warning state is also reflected in the endScript message result type.

Error

Indicates that the text will be entered as an error. The error state is also reflected in the endScript message result type.

Insert Code

Inserts code into the script based on the option you selected in the **Result** section, where `message` is the text you entered:

```
logInfo("message")
logWarning("message")
logError("message")
```

To open: If recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar and click the **Log Entry** tab. If editing, click the **Insert Recording into Active Functional Test scripts** button  on the Rational® Functional TesterHCL OneTest™ UI toolbar, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar, and click the **Log Entry** tab.

Logging page

You use the Logging page to set log and comparator options, such as preventing the script launch wizard from displaying on playback, displaying the log viewer after playback, and displaying a message before overwriting an existing log. You also use this page to indicate the type of log generated.

To access the logging page, click **Window > Preferences**. In the left pane, expand **Functional Test > Playback** and click **Logging**.



Note: For Microsoft Visual Studio, click **Tools > Options**. In the left pane, expand **Functional Test > Playback** and click **Logging**.

The Logging page contains the following options:

Don't show script launch wizard: When selected, prevents the script launch wizard from displaying each time you play back a script.

Display log viewer after script playback: When selected, this option displays the log after you play back a script. If the log type is HTML, the log opens in your default browser. If the log type is Text, the log opens in the Script Window of Rational® Functional TesterHCL OneTest™ UI. If the log type is XML, the log opens in your default browser.

Generally, the log file opens in the default browser that is associated with the html file extension in your computer. To view the html files in your desired browser, you can associate the html file extension with the specific browser. The file extension for different browsers are as follows:

- For Google Chrome, you must associate .html=ChromeHTML
- For Internet Explorer, you must associate .html=htmlfile
- For Firefox, you must associate .html=FirefoxHTML-308046B0AF4A39CB

Log screen snapshot for each action on the application: When selected, this option records a screen snapshot in the playback log against every action performed on the application.

Prompt before overwriting an existing log: When selected, this option prompts you before you overwrite a log.

Log the count of test objects created/unregistered at particular script line: When selected, this option logs these details:

- Number of objects created and unregistered at a specific script line
- Total number of objects created and unregistered per call script
- A cumulative number of test objects created and unregistered for the whole script during playback if Rational® Functional TesterHCL OneTest™ UI scripting methods have been used to return test objects.

Warning messages are also logged at the call script level and the main script level, if the number of test objects created exceeds the number of test objects unregistered, which would suggest the possibility of memory leaks during playback.

Log a screen snapshot when playback fails: When you select this option, it captures a screen snapshot at the time of the failure and stores it in the log. You must clear the check box to save storage space (172 KB per snapshot).

Log GUI actions on the application: When you select this option, it adds a detailed record of any GUI-related actions performed on the application (without a screen snapshot) to the playback log.

Log type: This option Indicates the type of log Rational® Functional TesterHCL OneTest™ UI generates to write results of script playback. The log types are as follows:

- **None**: Generates no log, if selected.
- **Text**: Displays the log in ASCII format in the Functional Test script window.
- **HTML**: Displays the log in HTML format in your default browser. The left pane in the HTML log contains three boxes: Failures, Warnings, and Verification Points. The list of items in each box help you navigate to a specific location in the log. You can select an item to quickly find important errors, warnings, and verification point results in the log. To do so, double-click an item in a list, and Rational® Functional TesterHCL OneTest™ UI scrolls to and displays the item in the log.

- **TPTP**: Displays a log using TPTP in the Functional Test script window.
- **XML**: Displays a log of XML data rendered in HTML format [using transformation and Cascaded Style Sheets] in your default browser.
- **Default**: Displays the unified report for the test scripts in the browser window. This is also the default option to generate result for Functional test scripts.
- **JSON**: Displays a log in JSON format in the Functional Test Script window. Each event in this log type is a separate JSON.



Note: The JSON log type is not supported in the integration of Rational® Functional TesterHCL OneTest™ UI with Visual Studio.

Use Default: Clear the check box to change the value in the **Log type** field. Select the check box to restore the default value.

Restore Defaults: Restores the default values on this page.

Apply: Saves your changes without closing the dialog box.

Merge test object into the test object map page

You use the Merge Test Objects into the Test Object Map page to merge multiple private or shared test object maps into a single shared test object map.

The Merge Test Object into the Test Object Map page lists all the scripts and test object maps in the current project.

To select multiple scripts and test object maps in the list, press and hold Ctrl.

The Merge Test Object into the Test Object Map page has the following controls:

Connect selected scripts with selected Test Object Map

Associates the new test object map with the scripts you chose. This control is only available if you select one or more scripts in the list. With this option, Rational® Functional TesterHCL OneTest™ UI updates the scripts you selected to point to the newly merged test object map. If you use Source control, Rational® Functional TesterHCL OneTest™ UI checks out the scripts and test object maps unreserved and leaves them checked out after the merge is complete.

Finish

Merges test objects from the maps and scripts you selected into the test object map.

To open: In the Projects view, right-click the test object map into which you want to merge test objects from other scripts and maps and click **Merge Objects into**.

Record Monitor Preferences dialog box

The Record Monitor Preferences dialog box enables you to modify Recording Monitor message options while in the Recording Monitor. You can add a timestamp to messages in the Record Monitor and select the text color for different types of messages.

Notes:

- You can also set preferences for the Record Monitor in the Recorder: Monitor tab on page 844 of the Preferences dialog box.
- Changes in this dialog box affect your user profile only.

The Record Monitor Preferences dialog box has the following tabs. Click the tab name below for information about how to use that tab.

Filter/Timestamp on page 1793

Use to add a timestamp to messages in the Record Monitor and to indicate the types of messages you want displayed there.

Text Color on page 1792

Use to select the text color for the different types of messages displayed in the Record Monitor. The colors you select stay in effect until you change them again.

To open: When recording, click the **Message Preferences** button  on the Record Monitor toolbar.

Message Text Color tab

You use the Message Text Color tab of the Monitor Options dialog box to select the text color for the different types of messages displayed in the Recording Monitor. The colors you select stay in effect until you change them again.

Note:

- You can also set preferences for the Recording Monitor in the Recorder: Monitor page of the Preferences dialog box on page 844.
- Changes in the Monitor Options dialog box affect your user profile only.

The **Message Text Color** tab has the following controls:

Select Message Level -- Select the type of message you want to set color for: **Error**, **Warning**, or **Information**.

Apply -- Makes the changes you have indicated without closing the Message Options dialog box.

Choose Text Color for Message Level -- Use any of the three tabs to select the color you want to use for the message:

- **Swatches** enables you to click the color you want to use. **Recent** displays the colors you selected while the dialog box was open.
- **HSB** enables you to select a color through its **H**ue, **S**aturation, and **B**rightness levels. The slider affects the hue. The resultant RGB values are also displayed.
- **RGB** enables you to enter a value or use a slider to select the levels of red, green, and blue. Values you select in this tab are also displayed in the **R**, **G**, and **B** fields in the **HSB** tab.

Preview -- Provides examples of the color you selected.

To open: When recording, click the **Monitor Message Preferences** button  on the Recording Monitor toolbar and click the **Message Text Color** tab.

Monitor Options tab

You use the Monitor Options tab of the Monitor Options dialog box to add a time stamp to messages in the Recording Monitor and to indicate the types of messages you want displayed there. You can also turn off the recording instructions automatically displayed at the top of the monitor window.

Notes:

- You can also set preferences for the Record Monitor in the Recorder: Monitor tab on page 844 of the Preferences dialog box.
- Changes in the Monitor Options dialog box affect your user profile only.

The Monitor Options tab has the following controls:

Show Getting Started Help

When selected, displays instructions on how to record a script using the buttons on the Recording toolbar at the top of the monitor window. The default is to display the Getting Started Help. If unchecked, the Getting Started Help does not display.

Include time stamp in the message

When selected, includes a time stamp, with the format *hh:mm:ss,* for each entry in the Recording Monitor. This option is off by default. When on, each message begins with the time stamp of when that action took place. The message goes out to seconds.

Select message types to be displayed

Select the messages you want to display in the Recording Monitor:

- Error
- Error, Warning
- Error, Warning, Information (default)

To open: When recording, click the **Monitor Message Preferences** button  on the Recording Monitor toolbar and click the **Monitor Options** tab.

Operating System page

You use the Operating System page to indicate the Foreground Lock Timeout setting for Windows® 98/Me and Windows® 2000 systems.

This page contains the following controls:

Foreground Lock Timeout

An important option for **Windows 98/Me**, **Windows 2000**, or later that sets the amount of time (in milliseconds) after user input, during which the operating system does not allow applications to force themselves into the foreground. To play back scripts, you must change this setting to **0**. However, this is a persistent setting and affects desktop behavior.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open: Click **Windows > Preferences**. In the left pane expand **Functional Test** and click **Operating System**.

To open: Click **Tools > Options**. In the left pane expand **IBM Functional Test** and click **Operating System**.

To open: Click **Tools > Options**. In the left pane expand **HCL Functional Test** and click **Operating System**.

Delays page

You use the Delays page to set delays during Functional Test script playback. These settings are useful to control the rate at which script commands are sent to the operating system.

The Delays page has the following controls:

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Delay before mouse up

Indicates, in seconds, the interval before sending a mouse-release event during playback.

Delay before mouse move

Indicates, in seconds, the interval before sending a mouse-move event during playback.

Delay before mouse down

Indicates, in seconds, the interval before sending a mouse-press event during playback.

Delay before performing Flex Test Object action

Indicates, in seconds, the wait before performing a Flex test object action during playback.

Delay before key up

Indicates, in seconds, the wait before sending a key-release event during playback.

Delay when hover

Indicates, in seconds, the duration of the wait for a Hover command, which takes no options.

Delay after top level window activates

Indicates, in seconds, the wait after making a new window active. This provides the application time to repaint the screen.

Delay before key down

Indicates, in seconds, the interval before sending a key-press event during playback.

Delay before performing Test Object action

Indicates, in seconds, the time the object waits before each UI action.

Restore Defaults

Restores the values on this page to customization file settings (if they exist) or to RATIONAL._FT.RFTCUST settings.

Apply

Saves your changes without closing the dialog box.

To open: Click **Window > Preferences.** In the left pane, expand **Functional Test**, expand **Playback,** and click **Delays**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Delays**.

Playback Monitor page

You use the Playback Monitor page to specify whether to display the playback monitor during playback.

The Playback Monitor page has the following controls:

Show monitor during playback

Displays the Playback Monitor during playback.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open, click **Window > Preferences**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Monitor**.click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **Monitor**.

General Playback page

You use the General Playback page to set script playback options, such as the amount of time Rational® Functional TesterHCL OneTest™ UI looks for an object and waits before trying to find an object again. You can also elect to skip all verification points in the script.

The General Playback page has the following controls:

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Show exception dialog

The exception dialog box is displayed if an exception occurs during playback.

Perform playback in interactive mode

To resolve common runtime situations dynamically.

Maximum time to attempt to find Test Object

The maximum amount of time, in seconds, that Rational® Functional TesterHCL OneTest™ UI attempts to find an object. See example on page 838.

Pause between attempts to find Test Object

Indicates, in seconds, how long Rational® Functional TesterHCL OneTest™ UI waits before trying to find an object again. See example on page 838.

Skip Verification Points

When selected, skips all verification points in the script.

Timeout used in waitForExistence( ) method

Indicates, in seconds, the maximum amount of additional time that Rational® Functional TesterHCL OneTest™ UI waits (after time specified in **Maximum time to attempt to find test object**) for an object. For example, this setting is useful when waiting for an application to open. The waitForExistence( ) method must be explicitly stated in the script.

Retry time used in waitForExistence( ) loop

Indicates, in seconds, the interval between attempts to find an object. If Rational® Functional TesterHCL OneTest™ UI does not find an object, it continues to try until the time specified in **Timeout used in waitForExistence( ) method** has expired.

Restore Defaults

Restores the default values on this page.

Apply

Saves the edits you made without closing the dialog box.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test** and click **Playback**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Playback** or **General.**

Related reference

Setting general playback preferences in test scripts on page 837

ScriptAssure page--Advanced

You use the ScriptAssure(TM) Advanced page to set thresholds for recognition scores, which Rational® Functional TesterHCL OneTest™ UI uses when searching for objects during script playback.

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

The ScriptAssure™ Advanced page has the following controls:

Maximum acceptable recognition score -- Indicates the maximum score an object can have to be recognized as a candidate. Objects with higher recognition scores are not considered as matches until the time specified in **Maximum time to attempt to find Test Object** has elapsed.

Last chance recognition score -- If Rational® Functional TesterHCL OneTest™ UI does not find a suitable match after the time specified in **Maximum time to attempt to find Test Object** has elapsed, indicates the maximum acceptable score an object must have to be recognized as a candidate. Objects with higher recognition scores are not considered.

Ambiguous recognition scores difference threshold -- Writes an AmbiguousRecognitionException to the log if the scores of top candidates differ by less than the value specified in this field. If Rational® Functional TesterHCL OneTest™ UI sees two objects as the same, the difference between their scores must be at least this value to prefer one object. You can override the exception by using an event handler in the script.

Warn if accepted score is greater than -- Writes a warning to the log if Rational® Functional TesterHCL OneTest™ UI accepts a candidate whose score is greater than or equal to the value in this field.

Standard -- Displays the Standard ScriptAssure(TM) preferences page, which enables you to use a slider to set the tolerance level from **Tolerant** to **Strict** for recognition levels and from **None** to **High** for warning levels.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves the edits you made without closing the dialog box.

Changes you make in this page are reflected in the ScriptAssure(TM) Page-Standard.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**. Click **Advanced**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**. Click **Advanced**.

ScriptAssure page-standard

During playback, Rational® Functional TesterHCL OneTest™ UI compares objects in the application-under-test with recognition properties in the test object map. You use the ScriptAssure(TM) Standard page to control object-matching sensitivity during playback. This feature enables you to successfully play back scripts when the application-under-test has been updated.

The ScriptAssure(TM) Standard page has the following controls:

Recognition Level -- Controls the level of recognition when identifying objects during script playback. To decrease tolerance for differences between the object in the application-under-test and the recognition properties, move the slider toward **Strict**. To increase the tolerance for differences, move the slider toward **Tolerant**.

- The maximum **Strict** setting indicates that objects must be an almost exact match. If only one important recognition property is wrong, Rational® Functional TesterHCL OneTest™ UI recognizes the object as a match after exhausting all other possibilities. An object with more than one wrong recognition property is not a match.
- The maximum **Tolerant** setting indicates that Rational® Functional TesterHCL OneTest™ UI selects an object with somewhat similar properties immediately.
- The default setting allows two important recognition properties to be wrong but still is a match if all other possibilities are exhausted. An object with more than two wrong recognition properties is not a match.

Warning Level -- Specifies when to be warned about differences between the object and the recognition properties. To increase the number of warnings, move the slider toward **High**. To decrease the number of warnings, move the slider toward **None**.

- The maximum **High** setting writes a warning to the test log of almost any difference. (Functional Test does not issue a warning when the difference is the browser.)
- The maximum **None** setting omits warnings to the test log of differences.
- With the default setting, Rational® Functional TesterHCL OneTest™ UI writes a warning to the test log whenever it finds a test object after the maximum time has elapsed during playback.

Advanced -- Displays the Advanced ScriptAssure Preferences page, which enables advanced users to set thresholds for recognition scores.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

Changes you make on this page are reflected in the ScriptAssure(TM) Page-Advanced.

To open: Click **Window > Preferences Tools > Options**. In the left pane, expand **Functional Test**, expand **Playback**, and click **ScriptAssure**.

Preferences dialog box

The Preferences dialog box contains pages that enable you to customize Rational® Functional TesterHCL OneTest™ UI in a number of different areas, such as settings for time options; colors for the Verification Point Editor, the Verification Point Comparator, and the Object Map Editor; highlight color for test objects; operating system; playback; delays; log; playback monitor; ScriptAssure(TM); recorder; recording monitor; and the workbench.

The Preferences dialog box has the following tabs. Click the tab name below for information about how to use that tab.

Functional Test **on page 828**

Use to easily increase or decrease all Rational® Functional TesterHCL OneTest™ UI time options.

Highlight **on page 831**

Use to specify how you want Rational® Functional TesterHCL OneTest™ UI to emphasize test objects when you select them in applications-under-test. You can specify color, width, speed, and time.

Operating System **on page 836**

Use to indicate values that are operating system-dependent.

Playback **on page 836**

Use to indicate settings for Functional Test script playback.

Delays **on page 839**

Use to indicate settings for delays during Functional Test script playback. These settings are useful to slow down the rate at which script commands are sent to the operating system.

Logging **on page 834**

Use to indicate log viewing and Comparator options.

Monitor **on page 840**

Use to indicate Playback Monitor settings.

ScriptAssure(TM) **on page 840**

Use the Standard Preferences page to control Rational® Functional TesterHCL OneTest™ UI's object-matching sensitivity during play back. Advanced users can use the Advanced Preferences page to set thresholds for recognition scores, which Rational® Functional TesterHCL OneTest™ UI uses when searching for objects during script playback.

Recorder **on page 843**

Use to indicate options for recording Functional Test scripts.

Monitor **on page 844**

Use to change colors without going to the Recording Monitor. Preference page buttons reflect the current settings in the Recording Monitor.

UI Color on page 830

Use to specify color settings for the Verification Point Editor, Verification Point Comparator, and the Object Map Editor.

Workbench on page 857

Use to indicate how you want the Workbench to behave while playing back, recording, and debugging Functional Test scripts.

To open: Click **Window > > Preferences** and in the left pane expand **Functional Test**.

Quick Find dialog box

You use the Quick Find dialog box to search a test object map for an object based on the property or value you enter.

The Quick Find dialog box has the following controls:

Find

Enter the string you want to search for or select one from the list. The list contains up to 10 of the most recently used search strings.

Match Case

Select to find only the text that matches the case of the string you entered in the **Find** field.

property

Select to search only properties for the text you entered.

value

Select to search only values for the text you entered.

either

Select to search properties or values for the text you entered.

When you click **OK**, Functional Test searches through the test object map for objects that meet the search criteria and highlights the first occurrence. Use the **Find: First**  , **Find: Previous**  , **Find: Next**  , or the **Find: Last**  buttons on the test object map toolbar to navigate between objects that meet the search criterion.

To open: From the test object map menu, select **Find > Quick Find** or from the test object map toolbar, click the **Find: Quick** button  .

Record a New Functional Test Script dialog box

You use the Record a New Functional Test Script dialog box to record a new test script. Rational® Functional TesterHCL OneTest™ UI includes in the new script import statements for files needed to compile and comments containing archiving information.

Rational® Functional TesterHCL OneTest™ UI uses the script name as the class name and sets up testMain, where you can add the commands you want to include in the script.

The Record a New Functional Test Script dialog box has the following controls:

Enter or select the folder

Either enter the appropriate path to the folder you want to use or use the navigation tools (**Home**  , **Back**  , and **Go Into** ) to select the path.

Script name

Enter the name you want to use for the new test script. Use Java™ file naming conventions.

Select Mode

Select the scripting mode in which you want to record the current script, either Simplified Scripting or Java Scripting.

Default

Makes the scripting mode you selected from the **Select Mode** list the default scripting mode.

Next >

Displays the Select Test Object Map wizard page on page 1834, which enables you to choose a private or a shared object map to use with the new test script.

Finish

Creates a new test script, using the default test object map.

Cancel

Closes the Record a New Functional Test Script dialog box without creating the new script.

To open: Click the **Record New Functional Test Script** button  on the product toolbar, or select **File > New > Functional Test Script Using Recorder**.

Related Topics for Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration:

Recording a Script on page 877

Creating a New Functional Test Script without Recording on page 879

Creating a New Functional Test Script without Recording on page 879

Recording a Script on page 902

Creating a New Functional Test Script without Recording on page 880

Recording interface

Every time you begin recording, the Rational® Functional TesterHCL OneTest™ UI window will automatically minimize when the Recording Monitor comes up. The monitor is always displayed while you are recording. You can minimize it if you don't want it to be visible on the screen, and you can resize it.

The monitor displays messages for every action performed during your recording session, such as starting and pausing the recording, starting an application or browser, clicks and all other actions upon an application, inserting verification points, inserting other items into the script, etc. For more information, see Rational® Functional TesterHCL OneTest™ UI Recording Monitor on page 1807.

The Record toolbar is nestled within the Rational® Functional TesterHCL OneTest™ UI Monitor toolbar when you bring it up. You can click **Display Toolbar Only**  , which hides the monitor and shows only the toolbar. Click **Display Monitor** to bring it back.

You use the Record toolbar to pause and stop recording, start applications, create verification points, and insert items into your script. For more information, see Recording Toolbar on page 1802.

The following dialog boxes make up the Rational® Functional TesterHCL OneTest™ UI recording interface:

- Verification Point and Action Wizard on page 1844
- Start Application dialog box on page 1837
- Script Support Functions dialog box on page 890
- Recorder Monitor Preferences Page on page 844

Recording toolbar

When you begin recording, the Recording toolbar comes up. You can use the Recording toolbar to pause and stop recording, start applications, create verification points, and insert items into your script. The Recording toolbar is nestled within the Monitor toolbar when you open the monitor.

You can display just the toolbar, by clicking the **Display Toolbar Only** button  . To re-dock it, click the **Display Monitor** button  .

For information on the Monitor and Monitor toolbar, see the Recording Monitor on page 1807.

The Recording toolbar contains the following buttons:

Stop Recording

Use to stop the recording. When recording stops, the Monitor is closed. Your script and object map are then written into your project directory. Also, the product window is restored and the script is displayed.

Pause Recording

Use to suspend recording. The Monitor remains in view.

Resume Recording

Use to continue recording.

Start Application

Use to start the application you want to test. It opens the Start Application dialog box, which is used to choose an application to start. For more information on starting applications, see Starting Your Test Applications on page 896 or the Start Application Dialog Box on page 1837.

Insert Verification Point or Action Command

Use to select an object and action. It opens the **Select an Object** tab of the Verification Point and Action Wizard, which contains the object selector tool. You select an object from there to start a verification point. For more information on selecting objects to create a verification point, see the Select an Object tab of the Verification Point and Action Wizard on page 1827.

Note: You can also use the **Insert Verification Point or Action Command** toolbar button itself to select an object. If you click the button and drag it off of the toolbar, the button will become the object selector tool from the **Select an Object** tab of the Verification Point or Action Wizard.

Insert Data Driven Commands

Use to select the test objects and actions that you want to data drive a test script on page 938.

Insert Script Support Commands

Opens the Script Support Functions dialog box. Use to call another script, insert a log entry, insert a timer, insert a sleep command (a delay), or insert a comment into your script. For more information see the Script Support Functions Dialog Box on page 890.

Display Help

Pauses your recording and displays the product Help in the Help Perspective. The Help contains all of the product user documentation. When the Help displays, the **Pause Recording** button becomes the **Resume Recording** button. Click the **Resume Recording** button to dismiss the Help window.

Display Toolbar Only

Displays just the Recording toolbar. Click the **Display Monitor** button to display the monitor and re-dock the Recording toolbar.

Manual test recording monitor

When you record a manual test script from Rational® Quality Manager, the manual test recording monitor is displayed.

Rational® Functional TesterHCL OneTest™ UI, version 8.2.2 supports the manual test script recorder feature in Rational® Quality Manager. When Rational® Quality Manager is integrated with Rational® Functional TesterHCL OneTest™ UI, you can record a manual test script in Rational® Quality Manager to test applications by using the

Rational® Functional TesterHCL OneTest™ UI recorder. The recorded test script is displayed in the Rational® Quality Manager editor and can be modified.

For more information about the manual test script recorder feature, see the Rational® Quality Manager Information Center.

Using the manual test recording monitor

When you initiate the recording of the manual test script from the Rational® Quality Manager computer, the Rational® Quality Manager interface minimizes and the manual test recording monitor is displayed on the Rational® Functional TesterHCL OneTest™ UI computer. You can open the application-under-test (AUT) and record your actions.

The monitor is displayed while you are recording, and is always the top window. The monitor displays messages for every action performed during the recording session, such as starting and pausing the recording, starting an application or browser, clicks and all other actions recorded against an application, inserting manual steps and inserting verification points into the script. The monitor also displays errors. You can minimize or resize the monitor, and you can also choose to see only the Recording toolbar by clicking the **Display Toolbar Only** button.

Note: You can insert data verification points or image verification points, but you cannot insert properties verification points, or get specific values for an object. For more information about verification points, see the related links at the bottom of this page.

When you stop recording, the monitor closes. In the Rational® Quality Manager browser, the **Manual Steps** list displays the recorded script actions.

The text in the recording monitor is similar to the sequence of commands that are recorded in the script. The text might contain additional information about errors that occurred during recording. The text is available while you have the monitor open. When you stop recording, the text is discarded and is not sent to Rational® Quality Manager. The text is not shown in the steps displayed in the **Manual Steps** list in Rational® Quality Manager. If you want to save the text, use the **Copy Selected Text** or **Save Monitor Text As** commands before you stop recording.

Manual test recording monitor toolbars

The recording monitor contains the **Recording** toolbar, which is the leftmost set of icons, and the **Monitor** toolbar, which is the rightmost set. The Recording toolbar is nested within the Monitor toolbar when you bring up the monitor. You can use the Recording toolbar to pause and stop recording, start applications, create a verification point, and insert items into your script. You can undock the Recording toolbar by clicking the **Display Toolbar Only** button  . The monitor will float so that it only displays the Recording toolbar. To dock the Recording toolbar again and show the monitor, click the **Display Monitor** button  . The display setting persists between recordings: if you only display the Recording toolbar, it is displayed this way the next time that you record.

Recording toolbar options

Table 1 lists the buttons available on the Recording toolbar and describes how to use them:

Table 61. Recording toolbar options for recording manual test scripts

Button	Action
Stop Recording	Stop the recording. When recording stops, the monitor closes. In the Rational® Quality Manager browser, the **Manual Steps** list displays the recorded script actions.
Pause Recording	Suspend recording. The monitor remains in view.
Resume Recording	Continue recording.
Start Application	Start the application that you want to test. The Start Application dialog box opens, in which you choose an application. For more information about starting applications, see the **Starting Your Test Applications** or **Start Application Dialog Box** links at the bottom of this page.
Insert Verification Point	Select an object and action. The **Select an Object** tab of the Verification Point Wizard is opens and displays the object selector tool from which you can select an object to start a verification point. You can insert data verification points or image verification points, but you cannot insert properties verification points, or get specific values for an object. For more information about selecting objects to create a verification point, see the **Select an Object tab of the Verification Point Wizard** link at the bottom of this page. For more information about verification points, see the related links at the bottom of this page. **Note:** You can also use the **Insert Verification Point** toolbar button itself to select an object. If you click the button and drag it off the toolbar, the button becomes the object selector tool from the **Select an Object** tab of the Verification Point wizard.
Insert Manual Step	Insert a manual step into the script. The Insert Manual Step dialog box opens, in which you can provide the name of the step.

Table 61. Recording toolbar options for recording manual test scripts

(continued)

Button	Action
Enable Image Capture	Enable the capturing of images during recording. When you enable this option, images are captured for the steps in the recording. If you do not want images to be captured, disable this option.

Monitor toolbar options

The **Monitor** toolbar is always embedded in the monitor. Table 2 lists the buttons on the **Monitor** toolbar, and how to use them:

Table 62. Monitor toolbar options

Button	Action
Clear All Monitor Text	Clear all text from the monitor. To save the text before you clear it, use the **Copy Selected Text** button or the **Save Monitor Text As** button on the Monitor toolbar. Clearing the monitor text does not affect your recorded script.
Save Monitor Text As	Save all the text in the monitor to a file. The Save As dialog box opens, in which you can specify a file name and location. The Save Monitor Text As command selects and saves all the text into a .txt file.
Monitor Message Preferences	The Monitor Options dialog box opens, in which you can set preferences for the Recording Monitor Getting Started Help, message time stamp, the message filter level, and the message color. When you click this button, the **Monitor options** and the **Message Text Color** tabs open. These options are available on these tabs: • **Show Getting Started Help**: Displays instructions at the top of the monitor window for how to record a script by using the buttons on the Recording toolbar. The default is to display the Getting Started Help. If you clear this option, the Getting Started Help is not displayed. • **Include time stamp in this message**: Designates whether each line in the monitor has a time stamp. This option is off by default. When on, each message begins with the time stamp of when that action took place. The time is specified including seconds. • **Select message types to display**: Specifies the message filter level to display in the record monitor. You can choose whether to display errors, warnings, or informational messages. The default is to display the most detailed level: all three types of messages.

Table 62. Monitor toolbar options

(continued)

Button	Action
	The **Message Text Color** tab specifies the text color of the different types of messages displayed in the recording monitor. The default colors are as follows: errors in red, warnings in yellow, and information in black. You can use this tab to change the display colors.

Related reference

Select an Object page of the Verification Point and Action Wizard on page 1827

Start application dialog box on page 1837

Record Monitor Preferences dialog box on page 1792

Related information

Verification points on page 904

Creating properties verification point on page 904

Creating a data verification point on page 907

Creating an image verification point on page 914

Verification Point Editor on page 917

Verification point comparator on page 923

Starting your test applications on page 896

Recording Monitor

When you begin recording, the Recording Monitor appears. The monitor is displayed while you are recording, and is always the top window. You can minimize it or resize it. You can also choose to see only the Recording toolbar by clicking the **Display Toolbar Only** button.

The monitor displays messages for every action performed during your recording session, such as starting and pausing the recording, starting an application or browser, clicks and all other actions recorded against an application, and inserting verification points and script support features into the script. It also displays errors. When the monitor displays actions recorded, some of it may look like code, but this text is for informational purposes only and is not meant to be pasted into a script.

When you stop recording, the monitor is closed. Your script and object map are then written into your project directory. Also, the Functional Test window is restored, and the script is displayed.

The text in the record monitor is similar to the sequence of commands that will be recorded in the script. It may contain additional information about errors that occurred during recording. The text is available while you have the monitor open. When you end recording, the text is discarded and is not written to a file. If you want to save the text, use the **Copy Selected Text** or **Save Monitor Text As** commands listed below, before you stop recording.

Toolbars

The recording monitor contains the **Recording** toolbar, which is the left-most set of icons, and the **Monitor** toolbar, which is the right-most set. The Recording toolbar is nested within the Monitor toolbar when you open the monitor. You can use the Recording toolbar to pause and stop recording, start applications, create a verification point, and insert items into your script. You can undock the Recording toolbar by clicking the **Display Toolbar Only** button  . The monitor will float such that it only displays the Recording toolbar. To re-dock the Recording toolbar and show the monitor, click the **Display Monitor** button  . The display setting is a sticky setting -- if you only display the Recording toolbar, it is displayed this way the next time you record.

For information on the **Recording** toolbar buttons, see Recording toolbar on page 1802.

The **Monitor** toolbar is always embedded in the monitor, and contains the following buttons:

 **Copy Selected Text** -- Copies text from the monitor. When you select text, the Copy Selected Text button becomes enabled. You can select some or all of the text within the monitor and click the Copy Selected Text button. The selected text is copied to your system clipboard. You can then paste the text into a new or existing file.

 **Clear All Monitor Text** -- Clears all text from the monitor. To save the text before you clear it, use the **Copy Selected Text** button  or the **Save Monitor Text As**  button on the Monitor toolbar. Clearing the monitor text does not affect your recorded script.

 **Save Monitor Text As** -- Saves all the text within the monitor to a file. It opens the Save As dialog box, which you can use to designate the name and location of the file. The Save Monitor Text As command selects and saves all the text into a .txt file.

 **Monitor Message Preferences** -- Opens the Monitor Options on page 1792 dialog box, which you can use to set preferences for the Recording Monitor Getting Started Help, message time stamp, the message filter level, and the message color.

When you click the **Monitor Message Preferences** button on the **Monitor** toolbar, two tabs appears: the **Monitor options** tab and the **Message Text Color** tab.

The **Monitor options** tab contains the following options:

Show Getting Started Help -- Displays instructions at the top of the monitor window on how to record a script using the buttons on the Recording toolbar. The default is to display the Getting Started Help. If unchecked, the Getting Started Help does not display.

Include time stamp in this message -- Designates whether you want each line in the monitor to have a time stamp. This option is off by default. When on, each message begins with the time stamp of when that action took place. The message goes out to seconds.

Select message types to display -- Specifies the message filter level you want to display in the Record Monitor. You can choose whether to display errors, warnings, or informational messages. The default is to display the most detailed level -- all three types of messages.

The **Message Text Color** tab specifies the text color of the different types of messages displayed in the Recording Monitor. The defaults are: errors in red, warnings in mustard yellow, and information in black. You can use this tab to change to different display colors.

Recorder Monitor page

You use the Recorder Monitor page to change settings in the Recording monitor, such as displaying the recorder toolbar or the Recorder Monitor, including a timestamp for messages, and selecting the types of messages you want to display and their colors.

The Monitor page has the following controls:

Display recorder toolbar only -- Displays the recorder toolbar or the full Recorder Monitor window.

Include time stamp in the message -- Includes a timestamp, of the format *hh:mm:ss*, for each entry in the Recorder Monitor.

Error message color -- Indicates the color of errors in the Monitor. Double-click the color to change it.

Warning message color -- Indicates the color of warnings in the Monitor. Double-click the color to change it.

Information message color -- Indicates the color of information messages in the Monitor. Double-click the color to change it.

Select message types to display -- Enables you to include or omit any three message types that appear in the Monitor:

- Error
- Error, Warning
- Error, Warning, Information

You can add a time to messages in the Record Monitor, and indicate the types of and colors used for the messages. To do so, click the **Message Preferences** button  on the Record Monitor toolbar while recording.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

General Recorder page

You use the General Recorder page to indicate options for recording Functional Test scripts, such as excluding an executable from being recorded and setting the delay before recording a mouse action or a keystroke. You can also select or clear the option to open the test object map if there is a new test object in the application.

This page has the following controls:

Note: In the **Use Default** field for each control, clear the check box to edit the value in the field or select the check box to restore the default value.

Record Test Object relative Verification Point -- When selected, the test object details are not recorded while inserting the verification points.

Maximum identifier length-- Option to control the maximum number of characters used in a Test Object identifier in the script.

Processes excluded from testing -- Enter the executable name of the process that you do not want to record. Separate multiple processes with commas.

Note: Only processes that are dynamically enabled must be added in this field.

Delay before recording a mouse action -- Sets the delay from the end of a mouse action to the command that appears in the recording monitor. A shorter delay may limit the quality of recording of actions that cause state changes. A value of 0.0 causes a delay until the beginning of the next action. If the value is lower than the double-click interval, Rational® Functional TesterHCL OneTest™ UI uses the double-click interval.

Delay before recording a keystroke -- Sets the delay from the last keystroke to the inputKeys command that appears in the recording monitor. A value of 0.0 causes a delay until the beginning of the next action.

Restore Defaults -- Restores the default values on this page.

Apply -- Saves your changes without closing the dialog box.

Bring up object map if there is new test object -- When selected, opens the test object map if a test object in the application is not currently in the map.

To open: Click **Window > Preferences**. In the left pane, expand **Functional Test** and click **Recorder**.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Recorder** or **General**.

Regular Expression Evaluator

The Regular Expression Evaluator is available from the test object map, Script Support Functions dialog box or the Verification Point Editor. You can use it to test a regular expression while editing an object property.

To use the Evaluator

From the test object map or Verification Point Editor, right-click a property value and convert it to a regular expression. When it is a regular expression, right-click again and click **Evaluate Regular Expression** to open the Evaluator.

The **Pattern** and **Match Against Value** fields contain the current value. To try an expression, edit the value in the **Pattern** field, and click **Evaluate**. The **Result** indicates whether the expression matched.

The Evaluator has the following controls:

Pattern - This field contains the current value when you convert to a regular expression. Use this field to edit the expression.

Perform Case Sensitive Match - This check box controls case sensitivity. It is set to "on" by default, and matching is case-sensitive. If you want matching to ignore case, clear this check box.

Match Against Value - This field contains the current value when you convert to a regular expression. The pattern you edit must match this expression because it reflects the property value in your application.

Result - When you click **Evaluate**, this field contains the result. If the patterns matches, you see **MATCH** in green letters. If the pattern does not match, you see **NOT A MATCH** in red letters. If you use an illegal character in your pattern, this field indicates an exception.

Matching String - If the pattern matches, you see the matching string. If you use an illegal character in your pattern, this field gives you the error. For example, "Missing close paren" indicates that you have an open parentheses but not a closing parentheses in the regular expression.

Evaluation Button - Edit the value in the **Pattern** field, and click this button to test it. Click **OK** to save the expression when you are satisfied with it.

Examples Button - For examples of regular expression syntax and use cases, see Regular Expression Examples.

For more information about using regular expressions and numeric ranges, see Replacing an Exact-Match Property with a Pattern.

Rename dialog box

You can use the Rename dialog box to rename a Functional Test script, test object map, log, or other Java™ files in a project, verification point or test object in the Script Explorer.

When you rename a script, Rational® Functional TesterHCL OneTest™ UI renames the script and all its related files, such as the helper script files, the private object map, and any verification point files. When you rename a test object map, Rational® Functional TesterHCL OneTest™ UI updates associated scripts with the new map name.

 **Note:**

- You need to change any callScript commands in scripts that reference the old script name; otherwise, Rational® Functional TesterHCL OneTest™ UI logs an error when you run those scripts.
- For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, a Rename command is available in the Navigator view that is part of the Eclipse Workbench. This Rename command only renames an individual file, not the collection of files that makes up a Functional Test script. Therefore, do not use the Rename command in the Navigator view to modify any Functional Test project assets.

Current name

The name of the test asset in the project that you selected.

New name

The new name of the test asset in the project (40 characters maximum). By convention, Java™ test script names usually start with an uppercase letter. Test asset names cannot contain the following characters: \ / : * ? " <> | ( ) or a space.

To open: In the Projects view, right-click the project item you want to rename and click **Rename**.

Renew all names in associated script (s)

You use the **Renew All Names in Associated Script(s)** dialog box to change all script test object names to more meaningful and precise names. Functional Test updates the names in associated scripts from the test object map.

The Renew All Names in Associated Script(s) dialog box has the following controls:

Associated Scripts -- Select the associated scripts that contains the names you want to update.

To open: From the Test Object Map menu, click **File > Renew All Names in Associated Script(s)**.

Renew Name in Associated Script(s) dialog box

You use the Renew Name in Associated Script(s) dialog box to change a script test object name to a more meaningful and precise name. Rational® Functional TesterHCL OneTest™ UI updates the name in associated script(s) in associated script(s) from the test object map.

The Renew Name in Associated Script(s) dialog box has the following controls:

Renew the name of map node *name* with -- Enter the script test object name used in the associated scripts.

Associated Scripts -- Select the associated scripts that contains the name you want to update.

To open: From the Test Object Map menu, click **Test Object > Renew Name in Associated Script(s)** or from the Test Object Map toolbar, click **Test Object: Renew Name in Associated Script(s)**.

Save File As dialog box

You use the Save File As dialog box to save a file under another name and location.

The Save File As dialog box has the following controls:

Enter or select the folder -- Use the navigation buttons (**Home**  , **Back**  , and **Go Into** ) to select the appropriate path to the folder you want to use. Use the navigation buttons (**Save In** arrow, **Current Folder**, **Up One Level**) to select the appropriate path to the folder you want to use.

File name -- Enter the name you want to use for the new file.

OK -- Saves the file using the new name.

Save -- Saves the file using the new name.

Unlike the Save Script As dialog box, the Save File As dialog box saves the current file and not any related files. Use the appropriate dialog box to get the results you want.

To open: Click **File> Save As** in the product menu. Click **File > Save *file* As**

Save Script As dialog box

You use the Save Script As dialog box to save a script under another name.

The Save Script As dialog box has the following controls:

Select a folder

Use the navigation buttons (**Home**  , **Back**  , and **Go Into** ) to select the appropriate path to the folder you want to use. Use the navigation buttons (**Save In** arrow, **Current Folder**, **Up One Level**) to select the appropriate path to the folder you want to use.

Script name

Enter the name you want to use for the new script. Use Java™ file naming conventions. Follow the class name requirements.

Finish

Saves the script and its related files, such as the helper script files, the private object map, and any verification point files using the new name.

Save

Saves the script and its related files, such as the helper script files, the private object map, and any verification point files using the new name.

To open: click **File> Save Script *scriptname* As**. click **File> Save *scriptname* As**

Rational® Functional TesterHCL OneTest™ UI Script Explorer

The Rational® Functional TesterHCL OneTest™ UI Script Explorer lists the script helper, helper superclass or helper base class, test dataset, verification points, and test objects for the current script.

In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, the Script Explorer is the right pane of the Functional Test perspective. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, the Script Explorer is the left pane of the VB.NET window.

Note: If you want to simplify the Visual Studio .NET user interface, you can click the **My Profile** tab on the **Start** page, select **Student Window Layout** in the Window Layout list, and select **(no filter)** in the Help Filter list. The Solution Explorer and Dynamic Help display on the left side and the Script Explorer displays on the right side of the Visual Studio .NET window.

The Verification Points folder contains all the verification points recorded for the script.

Double-clicking a verification point opens the Verification Point Editor on page 917.

The Test Objects folder contains a list of the test objects available for the script. Each test object in the list is preceded by an icon that indicates its role.

Double-clicking the test object map icon opens the test object map on page 1838.

Right-clicking a verification point, test object map, or test asset listed in the Script Explorer displays various menu options, listed here in alphabetical order.

Associate with dataset -- Associates a script with one or more datasets on page 954.

Change Superclass/Base Class -- For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, displays the Select helper superclass dialog box, which enables you to change the helper superclass for a script. For Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, displays the Select a Script Helper Base Class dialog box, which enables you to change the helper base class for a script.

Delete -- Removes the selected verification point, test object, or test dataset from the Script Explorer and inserts a problem marker in the Code Editor to indicate if the deleted verification point, test object, or test dataset is used.

Highlight -- Highlights the selected object in the application-under-test if it is running. For information, see Locating a Test Object in the Application.

Insert at Cursor -- Inserts the selected verification point or test object into the script at the cursor location.

Insert Test Object -- Opens the Select an Object dialog box, which enables you to select a test object to add to the test object map and the script.

Insert Verification Point -- Opens the **Select an Object** page of the Verification Point and Action Wizard, which enables you to select an object in your application to test.

Interface Summary -- Opens a Help topic that describes the selected test object, including the supported test data types and the default recognition properties.

Open -- If a verification point is selected, opens the Verification Point Editor. If a test object is selected, opens the test object map and highlights the selected object. If a test dataset is selected, displays the dataset editor.

Remove dataset Association -- Removes a private or public dataset association on page 954 with a test script.

Rename -- In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, opens the Rename on page 1811 dialog box. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, highlights the name, which enables you to type the new name.

To open: In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, the Script Explorer opens (by default) in the Functional Test Perspective. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, the Script Explorer opens (by default) in the Visual Studio .NET window.

Related Topics:

Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration -- About the Functional Test Perspective on page 46

Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration -- About the Visual Studio .NET Window on page 59

Script support functions

When you record or edit a functional test script, you can insert codes to add features into the test script by using the tabs in the **Script Support Functions** dialog. The following table lists the tab names and the features associated with each of the tabs:

Tab name	You can...
Call Script	Insert a statement to call another test script.
Log Entry	Insert a log message into the test script. During playback, this message is displayed in the log.
Timer	Insert or stop a timer in the current script. Timers remain in the running state until you stop them explicitly or close Rational® Functional TesterHCL OneTest™ UI.
Sleep	Insert a sleep command into the test script to delay the script for the time that you specify.
Comment	Insert a comment into the test script.
Clipboard	Insert system clipboard commands into the test script.

Note: You can open the **Script Support Functions** dialog in one of the following ways:

- You can click the **Insert Script Support Commands** button on the **Recording Monitor** toolbar when you record the test script.
- You can click the **Insert Recording into Active Functional Test Script** button on the **Functional Test** toolbar, and then click the **Insert Script Support Commands** button on the **Recording Monitor** toolbar when you edit the test script.

The following links provide more information:

- Calling a script from a functional test script on page 891
- Inserting a log message into a functional test script on page 892
- Using timers with functional test scripts on page 893
- Setting delays and sleep states for functional test script playback on page 894
- Inserting comments into a functional test script on page 895
- Inserting clipboard commands into a functional test script on page 895

Call Script tab: Script Support Functions dialog box

You use the Call Script tab to insert a callScript command into your Functional Test script.

The **Call Script** tab has the following controls:

Script Name

Lists all the scripts in the current project.

Insert Code

Inserts the `callScript(" scriptname")` code in the current script at the cursor location, where `scriptname` is the name you selected in the **Script Name** field.

dataset Iterator Count

Determines how many times a test script runs when you play back the test script. Specify the count according to the number of records you have in the dataset. Type or select the number of records in the dataset, or select **Iterate Until Done** to access all records in the dataset. For a call script, you can select **Use Current Record** to use the same record across the call script.

Note: You can also insert a callScript command from the Functional Test Projects view or script it manually.

To open: If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar and click the **Call Script** tab. If editing, click the **Insert Recording into Active Functional Test Script** button on the product toolbar, click the **Insert Script Support Commands** button on the Recording Monitor toolbar, and click the **Call Script** tab.

Comment tab: Script Support Functions dialog box

You use the Comment tab to insert a comment into a Functional Test script.

The **Comment** tab has the following controls:

Comment to add to the script

Enter text for the comment.

Note: Rational® Functional TesterHCL OneTest™ UI does not automatically wrap the text. Put returns after each line.

Insert Code

Inserts the text with the appropriate comment delimiter (//) preceding each line.

To open: When recording, click the **Insert Script Support Commands** button  on the Recording Monitor toolbar and click the **Comment** tab.

Log Entry tab: Script Support Functions dialog box

You use the Log Entry tab to insert a log message into a Functional Test scripts and indicate whether it is a message, warning, or an error. During playback, Rational® Functional TesterHCL OneTest™ UI inserts this information into the log.

The **Log Entry** tab has the following controls:

Message to write to the log

Enter the text you want to include in the log.

Result

Select the type of message you want to add to the log. The result type will be displayed in the log.

Information

Indicates that the text will be entered as a message.

Warning

Indicates that the text will be entered as a warning. The warning state is also reflected in the endScript message result type.

Error

Indicates that the text will be entered as an error. The error state is also reflected in the endScript message result type.

Insert Code

Inserts code into the script based on the option you selected in the **Result** section, where `message` is the text you entered:

```
logInfo("message")
logWarning("message")
logError("message")
```

To open: If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar and click the **Log Entry** tab. If editing, click the **Insert Recording into Active Functional Test scripts** button on the Rational® Functional TesterHCL OneTest™ UI toolbar, click the **Insert Script Support Commands** button on the Recording Monitor toolbar, and click the **Log Entry** tab.

Sleep tab: Script Support Functions dialog box

You use the Sleep tab to insert a sleep command into your Functional Test script to delay the script the length of time you specify.

The Sleep tab has the following controls:

(seconds) -- Enter the amount of time you want to delay Functional Test script execution in seconds. **Seconds** argument is floating point. For example, Sleep (0.5) sleeps for 1/2 second.

Insert Code -- Inserts the `sleep(seconds)` code at the cursor location in the script where *seconds* is the time you entered in the **(seconds)** field.

To open: If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar and click the **Sleep** tab. If editing, click the **Insert Recording into Active Functional Test Script** button on the product toolbar, click the **Insert Script Support Commands** button on the Recording toolbar, and click the **Sleep** tab.

Timer tab: Script Support Functions dialog box

You use the Timer tab to insert a timer into the current script and to stop the timer. Timers remain running until you stop them explicitly or exit Functional Test.

The Timer tab has the following controls:

Start Timer: Name -- Enter the name you want to use for the timer.

Insert Code -- Inserts the `timerStart("name")` code at the cursor location in the script, where *name* is the name you entered in **Start Timer: Name** field.

Stop Timer: Timers -- Select from the list the timer that you want to stop or enter the name of a timer that was used in another recording session. The list contains all the timers that have been started in the recording session.

Note: Do not insert a `timerStop` statement before the corresponding `timerStart` statement.

Insert Code -- Inserts the `timerStop("name")`code at the cursor location in the script where *name* is the name you selected in **Stop Timer: Timers** field.

To open: If recording, click the **Insert Script Support Commands** button on the Recording Monitor toolbar and click the **Timer** tab. If editing, click the **Insert Recording into Active Functional Test Script** button on the product toolbar, click the **Insert Script Support Commands** button on the Recording toolbar, and click the **Timer** tab.

Search for Java Environments dialog box

The Search for Java™ Environments dialog box is opened using the **Search** button in the Java™ Environments tab of the Enable Environments dialog box (the enabler). It is used to search your hard disk drive(s) for JREs to configure and/or enable them.

For information about the enabler, see the Java Environments tab on page 1785.

The Search for Java™ Environments dialog has the following options:

Quick Search can only be used on Windows® systems. It searches the registry for the Java™ environments, and is quicker than searching your hard disk drive(s).

Search All Drives scans all of your hard disk drives or partitions to locate all the Java™ environments on your system.

Note: You should not use the **Search All Drives** option to find JREs on Linux® or UNIX® systems. Instead use the **Search In** option and browse for the JRE.

Select **Search In** to browse to a specific disk drive or root directory to search.

After choosing one of the search mechanisms, click the **Search** button. This returns you to the enabler and fills in the **Java Environments** list.

Search for Web Browsers dialog box

The Search for Web Browsers dialog box is opened using the **Search** button in the Web Browsers tab of the Enable Environments dialog box (the enabler). It is used to search your hard disk drive(s) for web browsers to configure and/or enable them.

For information on the enabler, see the Web Browsers tab on page 1871.

The Search for Web Browsers dialog has the following options:

Select **Search All** to let the enabler locate all the browsers on your system. It will scan all of your hard disk drives or partitions, and list the browsers in the **Web Browsers** list of the Enable Environments dialog box. The list includes the full path name of each browser.

Note: You should not use the **Search All** option to find browsers on Linux® or UNIX® systems. Instead use the **Search In** option and browse for it.

Select **Search In** to browse to a specific drive or root directory to search.

After choosing one of the search mechanisms, click the **Search** button. This returns you to the enabler and fills in the **Web Browsers** list.

Select an Action page of the Verification Point and Action wizard

You open the Verification Point and Action Wizard with the **Insert Verification Point or Action Command** button on the **Recording** toolbar. Use the wizard to select objects to test in your application, and to select the types of tests to perform on them. This is how you record a verification point.

The Select an Object page on page 1827 is the first step. After you select an object, the **Select an Action** page appears.

Note: By default, the **After selecting an object advance to next page** check box is checked. If you clear the option, you must click **Next** to advance to the next page after you select an object.

On the Select an Action page, you choose an action to perform on the test object. The Select an Action page has the following four actions:

Perform Data Verification Point -- Insert a verification point Data command on page 1784 for the selected object to test the data in your object when you play back your script.

Perform Properties Verification Point -- Insert a Properties verification point command on page 1783 for the selected object to test the properties in your object when you play back your script.

Perform Image Verification Point -- Insert an Image verification point command on page 1849 for the selected object to test the image in your object when you play back your script.

Get a Specific Property Value -- Insert a getProperty command on page 1782 for the selected object to put a getProperty into your script and return the value during playback.

Wait for Selected Test Object -- Insert a waitForExistence command on page 1784 to set a wait state for an object during playback to check for its existence.

Select an Action page of the Verification Point and Action wizard (from Insert)

When you open the Verification Point and Action Wizard while inserting a verification point from the Script Explorer, the **Select an Object** page appears.

Use the wizard to select objects to test in your application, and to select the types of tests to perform on them.

Here is how you create a verification point. The Select an Object page on page 1827 is the first step. After you select an object, the **Select an Action** page appears.

Note: By default, the **After selecting an object advance to next page** check box is checked. If you clear the check box, you must click **Next** to advance to the next page after you select an object.

When inserting a verification point while not recording, the Select an Action page is where you choose which verification point to perform on the test object. The Select an Action page has the following verification points:

Perform Data Verification Point -- Insert a verification point Data command on page 1784 for the selected object to test the data in your object when you play back your script.

Perform Properties Verification Point -- Insert a Properties verification point command on page 1783 for the selected object to test the properties in your object when you play back your script.

Perform Image Verification Point -- Insert Image verification point command on page 1849 for the selected object to compare the image when you play back your script.

Select an Object dialog box

You use the Select an Object dialog box to select the object in your application you want to add to the test object map and a script.

About this task

When you select an object, Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

If you select the wrong object, or decide to add a different one, use any of the methods to select a different object. Once you click the **Finish** button, the object that is listed in the grid is the one that you will be adding.

The following three selection methods are available:

 Object Finder Tool on page 1821

 Object Browser on page 1822

 Delay Method on page 1822

 Object Finder Tool

About this task

This is the most common and direct method of selecting an object:

1. Select the **Object Finder** tool icon  and drag it over the object in your application that you want to select.

   Rational® Functional TesterHCL OneTest™ UI outlines the object with a highlight border.

2. Release the mouse button.

   The object is selected and Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

Object Browser

About this task

Use the Object Browser method to browse for the object you want to add to the test object map. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level, Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application.

1. Browse the object tree to find the object you want to add to the test object map.
2. Click the object to select it.

   Rational® Functional TesterHCL OneTest™ UI lists the object's recognition properties in the grid at the bottom of the Select an Object page.

Delay Method

About this task

Use the Delay method to select pop-up objects, such as menus. This method uses the **Object Finder** tool, but enables you to set a delay, which gives you time to get to an object that requires clicking on other objects first.

1. In the **Seconds before selection** field, enter the number of seconds you want to delay before Rational® Functional TesterHCL OneTest™ UI attempts to find the object (the default is 10 seconds).
2. Click the **Object Finder** tool icon .
3. Go to the application and find the object you want to select.
   Anything you do during the delay period is not recorded, which enables you to dig for objects if necessary.

   When the timer runs out, Rational® Functional TesterHCL OneTest™ UI selects the object under the cursor, outlines the object with a highlight border, and displays its Recognition properties in the grid at the bottom of the Select an Object page.

   Note: In order for delayed location of the objects to play back correctly, the object must be exposed by actions in the script. If the object is not exposed, an Object Not Found exception is thrown during playback.

Results

Object Recognition Properties Grid

When you select an object using any of the above methods, its Recognition properties are listed in the grid at the bottom of the tab. The Recognition properties are determined by the object's proxy. For example, a button object has three recognition properties: label, .class, and .classIndex. The grid lists the name and value of your specific object's recognition properties. You can use that information to confirm that you selected the correct object.

To open: From the product toolbar, click the **Insert Test Object into Active Functional Test script** button  . You can also right-click the **Test Objects** folder in the Script Explorer and select **Insert Test Object**.

Select an Object page of the Insert a GUI Object into the Object Map wizard

You can use the Select an Object page of the Insert a GUI Object into the Object Map dialog box to select the object in your application you want to add to the test object map.

About this task

When you select an object, Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

Object Recognition Properties Grid

When you select an object using any of the three methods, its recognition properties are listed in the grid at the bottom of the Select an Object page. The Recognition properties are determined by the object's proxy. For example, a "button" object has three Recognition properties: label, .class, and .classIndex. The grid lists the name and value of your specific object's Recognition properties. You can use the Recognition Properties information to confirm that you selected the correct object.

To open: From the Test Object Map menu, click **Test Object > Insert Object(s)**. From the Test Object Map toolbar, click the **Test Object: Insert Object(s)** button  . You can also right-click in the test object map and click **Insert Test Object(s)**.

If you select the wrong object, or decide to add a different one, use any of the methods to select a different object. Once you click the **Finish** button, the object that is listed in the grid is the one that you will be adding.

The following three selection methods are available:

 Object Finder Tool on page 1823

 Object Browser on page 1824

 Delay Method on page 1824

Object Finder Tool

About this task

Using the Object Finder tool is the most common and direct method of selecting an object:

1. Select the **Object Finder** tool icon  and drag it over the object in your application that you want to select.

   Rational® Functional TesterHCL OneTest™ UI outlines the object with a highlight border.

2. Release the mouse button.

   The object is selected and Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

Results

Select the **After selecting an object advance to next page** option to go directly to the next page after you select the object. Clear the advance to next page option to remain on the Select an Object page to see the object recognition properties after you select the object.

Object Browser

About this task

Use the Object Browser method to browse for the object you want to add to the test object map. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level, Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application.

1. Browse the object tree to find the object you want to add to the test object map.
2. Click the object to select it.

   Rational® Functional TesterHCL OneTest™ UI lists the object's recognition properties in the grid of the Select an Object page.

Delay Method

About this task

Use the Delay method to select pop-up objects, such as menus. The Delay method uses the Object Finder tool, but enables you to set a delay, which gives you time to get to an object that requires clicking on other objects first.

1. In the **Seconds before selection** field, enter the number of seconds you want to delay before Rational® Functional TesterHCL OneTest™ UI attempts to find the object (the default is 10 seconds).
2. Click the **Object Finder** tool icon  .
3. Go to the application and find the object you want to select.
   Anything you do during the delay period is not recorded, which enables you to "dig" for objects if necessary.

When the timer runs out, Rational® Functional TesterHCL OneTest™ UI selects the object under the cursor, outlines the object with a highlight border, and displays its recognition properties in the grid at the bottom of the Select an Object page.

Note: In order for delayed location of the objects to play back correctly, the object must be exposed by actions in the script. If the object is not exposed, an Object Not Found exception is thrown during playback.

Results

Select the **After selecting an object advance to next page** option to go directly to the next page after you select the object. Clear the advance to next page option to remain on the Select an Object page to see the object recognition properties after you select the object.

Select an Object page of the Update Recognition Properties wizard

You use this page of the Update Recognition Properties wizard to select the object in your application for which you want to update recognition properties.

About this task

When you select an object, Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

If you select the wrong object, or decide to add a different one, use any of the methods to select a different object. The following three selection methods are available:

Object Finder Tool on page 1825

Object Browser on page 1826

Delay Method on page 1826

Object Finder Tool

Before you begin

About this task

The Object Finder tool is the most common and direct method of selecting an object:

1. Select the **Object Finder** tool icon and drag it over the object in your application that you want to update.

   Rational® Functional TesterHCL OneTest™ UI outlines the object with a highlight border.

2. Release the mouse button.

   The object is selected and Rational® Functional TesterHCL OneTest™ UI lists its recognition properties in the grid at the bottom of the Select an Object page.

Results

Select **After selecting an object advance to next page** option to go directly to the next page after you select the object. Clear the advance to next page option to remain on the Select an Object page to see the object recognition properties after you select the object.

 Object Browser

About this task

Use the Object Browser method to browse for the object you want to update. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level, Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application.

1. Browse the object tree to find the object you want to update.
2. Click the object to select it.

   Rational® Functional TesterHCL OneTest™ UI lists the object's recognition properties in the grid at the bottom of the Select an Object page.

 Delay Method

About this task

Use the Delay method to select pop-up objects, such as menus. The Delay method uses the **Object Finder** tool, but enables you to set a delay, which gives you time to get to an object that requires clicking on other objects first.

1. In the **Seconds before selection** field, enter the number of seconds you want to delay before Rational® Functional TesterHCL OneTest™ UI attempts to find the object (the default is 10 seconds).
2. Click the **Object Finder** tool icon  .
3. Go to the application and find the object you want to select.
   Anything you do during the delay period is not recorded, which enables you to dig for objects if necessary.

   When the timer runs out, Rational® Functional TesterHCL OneTest™ UI selects the object under the cursor, outlines the object with a highlight border, and displays its recognition properties in the grid at the bottom of the Select an Object page.

Results

Select **After selecting an object advance to next page** option to go directly to the next page after you select the object. Clear the advance to next page option to remain on the Select an Object page to see the object recognition properties after you select the object.

Object Recognition Properties Grid

When you select an object using any of the above methods, its recognition properties are listed in the grid at the bottom of the Select an Object page. The recognition properties are determined by the object's proxy. For example, a button object has three recognition properties: label, .class, and .classIndex. This grid lists the name and value of your specific object's recognition properties. You can use that information to confirm that you selected the correct object.

To open: From the test object map menu, select **Test Object > Update Recognition Properties**. From the Test Object Map toolbar, click the **Test Object: Update Recognition Properties** button  . You can also right-click in the test object map and select **Update Recognition Properties**.

Select an Object page of the Verification Point and Action Wizard

The Verification Point and Action Wizard is opened with the **Insert Verification Point or Action Command** button  on the **Recording** toolbar or the product toolbar. It is used to select objects or images to test in your application, and to select the types of tests to perform on them.

The **Select an Object** page is the first step. You use one of the selection methods on this page to select the object in your application you want to perform a test on. When the object is selected, its recognition properties are listed in the grid at the bottom of the page.

To perform an image verification test, use the **Capture Screen Image** tool to select the image or use the **Object Finder** tool to select the object and create an image verification point.

If you select the wrong object, or decide to test a different one, use any of the methods and select a different object. It will then be shown in the grid. Once you advance past this first page by clicking the **Next** button, the object that is listed in the grid is the one that you will be testing. After you select a test object, you'll select an action in the next page of the wizard. Once you have moved on, you can always click the **Back** button to select a different object.

The following three selection methods are available:

Object Finder Tool

Use this tool to select an object and all descendents of the object, select one object, or select an object and the immediate children of an object.

This is the most common and direct method of selecting an object. Grab the Object Finder tool icon with your mouse and the cursor turns into the tool. Drag it over the object in your application that you want to select. You'll see that the object is highlighted and the object name is displayed. When you release the mouse button, the object is selected, and its recognition properties are listed in the grid.

Note that you can also use the **Insert Verification Point or Action Command button**  **on the Recording** toolbar directly to select an object. If you click it and drag it off of the toolbar, it will become the object selector tool from this page of the wizard.

If the **After selecting an object advance to next page** option is selected, you'll go directly to the next page of the wizard after you select the object. Clear this option if you want to remain on this page to see the object recognition properties after selecting the object.

Object Browser

Use this method to browse for the object that you want to select. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level,Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application. It is a dynamic view of the currently available objects.

Using this method, you browse for your object. The browser displays a hierarchical tree of objects in your system that are testable. The top level shows any applications you have running, and under each one is the object hierarchy within that application. It is a dynamic view of the currently available objects. Browse the object tree till you find the object, then click it. That will select it, and its recognition properties will be listed in the grid.

Delay Method

Use this method to select pop-up objects, such as menus. This method uses the Object Finder tool, but enables you to set a delay, which gives you time to get to an object that requires clicking on other objects first.

This uses the Object Finder tool, but with a delay that you set. The delay gives you time to get to an object that requires clicking on other objects first, such as a menu command. Set the number of seconds (the default is 10), then click the tool icon. Move your mouse to hover over your application until you get to the object you want to select. Anything you do during that delay period is not recorded. This allows you to "dig" for objects if necessary. For example, you might click with your mouse to cause a menu to open. The timer counts down, and when it runs out the object under the cursor is selected, and its recognition properties will be listed in the grid.

Note: In order for the delayed location of objects to play back correctly, the object must be exposed by actions in the script. If the object is not exposed, an Object Not Found Exception is thrown during playback.

If the **After selecting an object advance to next page** option is selected, you'll go directly to the next page of the wizard after you select the object. Clear this option if you want to remain on this page to see the object recognition properties after selecting the object.

Object recognition properties grid

When the object is selected by any of the above methods, its recognition properties are listed in the grid at the bottom of the page. The recognition properties are determined by the object's proxy. For example, a "button" object has three recognition properties: label, .class, and .classIndex. This grid will list the name and value of your specific object's

recognition properties. You can use that information to confirm that you selected the correct object. If no information is listed, the object is not testable or the environment may not be enabled.

Capture Screen Image

To perform image verification test, use the **Capture Screen Image**  tool to capture the screen. This tool captures the full image of the screen.

Your next step

After you select an object using one of the methods listed above, click the **Next** button to choose an action to perform on the object. These include creating a data verification point, creating a properties verification point, creating an image verification point, getting a single property value, or setting a wait state on an object. For more information on the actions, see the Select an Action Page on page 1820.

Select Helper Superclass/Select a Script Helper Base Class dialog box

In Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, use the Select Helper Superclass dialog box to select a helper superclass to add to the script. In Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, use the Select a Script Helper Base Class dialog box to select a helper base class to add to the script.

The Select Helper Superclass/Select a Script Helper Base Class dialog box has the following controls:

Select default helper superclass for the script (Rational® Functional TesterHCL OneTest™ UI Eclipse Integration only) -- Enter a partial or complete fully qualified class name of your custom helper superclass in this field, which enables you to display the class names in the Matching Types list. Note that your helper superclass must extend RationalTestScript. The asterisk (*) indicates that all the classes that are not scripts appear in the Matching Types list.

Matching types -- Displays a list of classes. The default class is RationalTestScript. You can change the default helper superclass on page 887 for a project on the Functional Test Project Properties Page on page 1770 or for an individual script on the Functional Test Script Properties Page on page 1773.

Qualifier -- Displays the location of the selected class.

Script Helper Base Class (Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration only) -- Displays the base class you select from the list of base classes, which is located above this field. You can change the default script helper base class for a script on the Functional Test Project Properties Page or for an individual script on the Functional Test Script Properties Page.

To open: Click the **Record a Functional Test Script** button on the product toolbar, and click **Next**. On the Select Script Assets page, click **Browse** for Helper Superclass.

Click the **New Functional Test Script Using Recorder** button on the product toolbar, and click **Open**. On the Select Script Assets page, click **Browse** for Helper base class.

Select Items to Export page

You use the Select Items to Export page of the Export wizard to choose the project items such as scripts, test object maps, Java™ files or Visual Basic files, and datasets that you want to export to another Functional Test project.

Select items to export -- Lists all scripts and associated files, test object maps, datasets, and Java™ files or Visual Basic files. Select the check boxes of the items you want to export and clear those you do not want to export.

When you export a script, Rational® Functional TesterHCL OneTest™ UI includes any necessary files, such as shared test object maps, even though they are not checked in the **Select items to export** field.

Specify the export destination -- Enter the name or navigate to the data transfer file, which is the file you want to use to export the project items you have selected. If the file does not exist, Rational® Functional TesterHCL OneTest™ UI creates it using the name you enter and appends a .rftjdtr .rftvdtr extension.

Finish -- Compresses a copy of the files into a data transfer file with the name you specified and appends .rftjdtr .rftvdtrextension.

To open: Select the project in the Project view, click **File > Export**, select **Functional Test Project Items** and click **Next**. Right-click the project in the Solution Explorer and click **Export**.

Select Items to Overwrite page

Rational® Functional TesterHCL OneTest™ UI displays the Select Items to Overwrite page if the project already contains any of the items you are importing. You use the page to select the check boxes of the items that you want to overwrite in the project or clear the items that you do not want overwritten.

Select All -- Overwrite all items in the project with items listed on the page.

Deselect All -- Clear all items in the list so that none of them are overwritten.

Finish -- Add all project items from the data transfer file to the project, overwriting only those you specified.

- When you overwrite a script, Rational® Functional TesterHCL OneTest™ UI overwrites all the necessary files associated with the script, such as the test object map and the dataset. Other files in the project associated with the overwritten script, such as verification points, are deleted.

To open: In the Import Project Items page, click **Next**. Rational® Functional TesterHCL OneTest™ UI displays the Select items to overwrite page if the project already contains any of the assets you are importing.

Select object options page

You can use the Select Object Options page to select whether you want to add a selected object, a selected object and its immediate children, or all descendents of the selected object in a dataset table.

Selected object

Displays the name of the object that you selected.

Just the selected object

Click to include only the selected object in a dataset table.

Include the immediate children of the selected object

Click to include only the immediate children of the selected object in a dataset table.

Include all descendents of the selected object

Click to include all descendents of the selected object, whether displayed or not in a dataset table.

To open: Start recording a test script on page 877, click **Insert Data Driven Commands** (), on the Recording toolbar. Click **Use selection wizard to select test object**. Select an object using the Test Object Browser or Drag Hand Selection method. Click **Next**.

Note: The **Drag Hand Selection** method is not available on Linux environments such as Ubuntu and Red Hat Enterprise Linux (RHEL). You must use the **Test Object Browser** method on Linux environments.

Select Log page

Use the Select Log page to select a log and specify whether to have the Script Launch wizard open when you run a script.

The Select Log page has these controls:

Log Name for *script name*: Click to list all the log names for the current project. Enter or select a log name to create a log that displays your playback results.

Don't show this wizard again: Select to prevent the Script Launch wizard from starting each time you run a test script.

By default, the Script Launch wizard starts. To prevent the Script Launch wizard from starting when you run a test script:

1. Click **Windows > Preferences**
2. Click **Functional Test > Playback > Logging**.
3. On the **Logging options** page, select the **Don't show script launch wizard** check box.

1. Click **Tools > Options**.
2. Click **Functional Test > Playback > Logging**.
3. On the **Logging options** page, select the **Don't show script launch wizard** check box.

Enable handling of unexpected windows: If the unexpected window handling feature has been enabled for all scripts in the Preferences dialog box, this check box is selected. Clear the check box if you do not want to enable the feature for the script you are running.

If the unexpected window handling feature has not been enabled for all scripts in the Preferences dialog box, this check box is not selected. Select the check box if you want to enable the feature for the script you are running. Actions that have been configured for specific controls on unexpected windows in the Configure Handling of Unexpected Windows dialog box are performed.

Enable script find if scoring find fails: If the dynamic find feature has been enabled for all scripts in the Preferences dialog box, this check box is selected. Clear the check box if you do not want to enable the feature for the script you are running. The dynamic find feature enables Rational® Functional TesterHCL OneTest™ UI to locate test objects in the application-under-test whose hierarchical position may have been altered from the position in the test object map, ensuring that playback does not fail.

If the dynamic find feature has not been enabled for all scripts in the Preferences dialog box, this check box is not selected. Select the check box if you want to enable the feature for the script you are running.

Enable script find if scoring find fails: Select this check box to enable the dynamic find feature for the script that you are running. Clear the check box to disable the feature for the script. The dynamic find feature enables Rational® Functional TesterHCL OneTest™ UI to locate test objects in the application under test whose hierarchical position might have been changed from the position in the test object map, helping to ensure that playback does not fail.

To open the Select Log page:

- Click a script and click the **Run Functional Test Script** button  in the product toolbar.
- In the Projects view, right-click a script, and click **Run**.
- In the Projects view, click a script, and from the product menu click **Script > Run**.

To open the Select Log page:

- In the Solution Explorer, click a script, and click the **Run Functional Test Script** push button  in the product toolbar.
- In the Solution Explorer, right-click a script, and click **Run**.
- In the Solution Explorer, click a script, and from the Functional Test menu, click **Script > Run**.

Select Object Options page of the Insert GUI Object into the Object Map wizard

You use the Select Object Options page of the Insert a GUI Object into the Object Map dialog box to indicate that you want to add just one object, just the one object and its siblings, or all objects on the window.

The Select Object Options page has the following controls:

Just the selected object -- Includes in the test object map only the object you selected from the Select an Object page in the Insert a GUI Object into the Object Map dialog box.

Include the siblings of the selected object -- Includes the children of the immediate parent.

Include all available objects on this window -- Includes in the test object map the object you selected on the Select an Object page in the Insert a GUI Object into the Object Map dialog box and all the other objects on the window, whether displayed or not.

Back -- Returns to the **Select an Object** page in the Insert a GUI Object into the Object Map dialog box.

Finish -- Creates and displays the new test object map with the objects you selected.

Cancel -- Closes the Insert a GUI Object into the Object Map dialog box without adding the objects you selected.

Help -- Displays information about the Insert a GUI Object into the Object Map dialog box.

To open: When adding test objects to an object map, click **Next** on the Select an Object page in the Insert a GUI Object into the Object map dialog box.

Select object to data drive page

You can use the Select Object to Data Drive page to select an object that you want to data drive in your application-under-test. After you select an object, the objects recognition properties appear in the Object Recognition Properties grid at the bottom of the page.

Selection method

Click one of the selection methods from the list to select an object to data drive.

Drag Hand Selection

This is the most common and direct method of selecting an object. Click and drag the Object Finder tool over an object that you want to data drive, and then release the mouse button.

 **Object Finder icon**

Click and drag over an object that you want to data drive.

After selecting an object advance to next page

Clear or select to go to the next page automatically after you select an object.

 **Note:** The **Drag Hand Selection** method is not available on Linux environments such as Ubuntu and Red Hat Enterprise Linux (RHEL). You must use the **Test Object Browser** method on Linux environments.

Test Object Browser

Use this selection method to browse for the object you want to data drive. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level, Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application. The object hierarchy is

a dynamic view of the currently available objects. Browse the object tree until you find the object you want, and then click it. The object recognition properties appear in the grid.

Object Recognition Properties Grid

When you select an object using any of the selection methods, the object recognition properties are listed in the grid at the bottom of the tab. The recognition properties are determined by the object's proxy. For example, a "button" object has three recognition properties: label, .class, and .classIndex. This grid lists the name and value of your specific object's recognition properties. You can use that information to confirm that you selected the correct object.

To open: Click **Insert Data Drive Commands** () on the **Recording** toolbar. Click **Use selection wizard to select test objects**.

Select Script Assets page

You use the Select Script Assets page to select the type of object map, helper superclass or helper base class, test dataset, and dataset iterator class you want to use with the Functional Test script you are creating.

The Select Script Assets page has the following controls:

Test Object Map -- Opens **Private Test Object Map** to indicate that the script test object map is private or displays the name of the shared test object map.

Helper Superclass -- Enter the fully qualified class name of your custom helper superclass in this field. Note that your helper superclass must extend RationalTestScript.

Helper Base Class -- Enter the fully qualified class name of your custom helper base class in this field. Note that your helper base class must extend RationalTestScript.

If you change your superclass or base class and reset it to RationalTestScript, you can either type RationalTestScript in the superclass or base class field or clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

Test dataset -- Click **Browse** to change the dataset associated with a script.

dataset Record Selection Order -- Determines how a test script accesses records in its associated dataset when you play back the test script. Click the **dataset Record Selection Order** arrow to change the dataset record selection order.

Types of dataset record selection orders:

Sequential -- Makes a test script access records in the dataset in the order that they appear in the dataset. This is the default dataset record selection order.

Random -- Makes a test script access records in the dataset randomly. A random dataset record selection order accesses every record in the dataset once.

Set as test asset default for new scripts in this project -- Sets the test object map or the helper superclass as the default. Functional Test uses this default when you create a Functional Test script in the current project.

- **Test Object Map** -- Sets the map named in the Test Object Map field as the default. To change the default test object map in the Projects view, right-click a test object map and select **Set Test Object Map as Default**. To remove the default designation, right-click and click **Clear As Project Default**.
- **Helper Superclass** -- Sets the fully qualified class name of your custom helper superclass as the default.
- **Helper Base Class** -- Sets the fully qualified class name of your custom helper base class as the default.

Back -- Returns to the previous page.

Finish -- Creates the script, using the test object map you selected, and adds it to the Functional Test Projects view.

To open: When recording, click **Next** in the Record a Functional Test Script dialog box. When creating a new script without recording, click **Next** in the Create a New Functional Test Script dialog box.

s

Select Script to Play Back/Select Script to Debug dialog box

You can use the **Select Script to Play Back** or **Select Script to Debug** dialog box to select the Functional Test script that you want to play back or debug.

You can only play back or debug one script at a time.

Script Name

Type the name of the script that you want to play back or debug, or select a script from the list.

Finish

Click to play back or debug the script.

Set Active Find Criteria dialog box

You use the Set Active Find Criteria dialog box to select the filter you want to use for searching the test object map, to create a new filter, and to edit or delete an existing filter.

The Set Active Find Criteria dialog has the following controls:

Find Filter Names -- Lists the names of all the active find criteria available. The default is Test Object is New, which searches the test object map for all New objects.

Find Filter -- Lists all the find criteria, their properties, and relationships.

Create -- Displays the Define Find Filter Properties on page 1752 dialog box, which enables you to specify properties for a new set of find criteria.

Edit -- Displays the Define Find Filter Properties on page 1752 dialog box, which enables you to change properties for the selected find criteria.

Delete -- Deletes the selected find filter from the list.

OK -- Runs the selected filter.

Cancel -- Closes the dialog box without making any changes.

To open: From the test object map menu, select **Find > Find by Filters**; from the test object map toolbar, click the Find: Filters button .

Set Description Property for Selected Test Object dialog box

You use the Set Description Property for Selected Test Object dialog box to add descriptive text to the **Administrative property set** tab for the object. Rational® Functional TesterHCL OneTest™ UI displays the description when you place the cursor over the object name in a script.

Note: Rational® Functional TesterHCL OneTest™ UI places the description in the helper class. The description is not inserted until the helper class file is regenerated, which occurs automatically when you record, playback, or update recognition properties. To manually regenerate the helper class file, in the Rational® Functional TesterHCL OneTest™ UI menu, click **Script > Update Script Helper**.

Test Object Description for *object*, enter the text you want to use for the description property for the *object* named.

If a description property already exists for the test object, Functional Test displays it in the text box, which you can edit as necessary.

To open: Select an object in the test object map and either from the test object map menu, click **Test Object > Description Property** or from the test object map toolbar click the **Test Object: Description** button . You can also right-click a test object and click **Description Property**.

Set Highlight Window Preferences dialog box

You use the Set Highlight Window Preferences dialog box to specify how you wantRational® Functional TesterHCL OneTest™ UI to emphasize test objects in applications-under-test when you select them in a test object map or in the Script Explorer. These settings also control howRational® Functional TesterHCL OneTest™ UI highlights objects you select with various other wizards and dialog boxes, such as Verification Point and Action Wizard and the Insert a GUI Object into the Object Map dialog box.

Notes:

- You can also set these highlight preferences in the Highlight on page 831 page.
- Changes in this dialog box affect your user profile only.

The Set Highlight Window Preferences dialog box has the following controls:

Color -- Click to display a color selection palette from which you can click the color you want to use to indicate selected test objects. The field displays the RGB values of the color currently in use. Red (RGB=255,0,0) is the default.

Border Width (in pixels) -- Move the slider from **Thin** to **Wide** to designate the width of the border around the selected object.

Flash Speed -- Move the slider from **Slow** to **Fast** to designate the rate at which you want the border around a selected object to flash when selected.

Display Time -- Move the slider from **Short** to **Long** to designate the length of time you want the border highlighted.

To open: In the test object map menu, click **Preferences > Highlight**.

Start application dialog box

You can open the Start Application dialog box with the **Start Application** button on the Recording toolbar to start your test application while you are recording.Rational® Functional TesterHCL OneTest™ UI inserts a startApp command into your script.

Application Name

Click the arrow to see the list of applications you can test. Java™ applications will be indicated as "Application name - java", HTML applications will be indicated as "Application name - htm", and executables or batch files as "Application name - executable". Select the application you want to open and click **OK**.

Rational® Functional TesterHCL OneTest™ UI is shipped with several sample applications that will appear in the Start Application dialog box. For example, "ClassicsJavaA - java" is used in theRational® Functional TesterHCL OneTest™ UI Tutorial. To run the tutorial, see theRational® Functional TesterHCL OneTest™ UI tutorials.

You can configure your own applications so that they will appear in this list and you can start them using this dialog box. You add your applications by clicking the **Edit Applications List** button.

Edit Applications List Button

When you click the **Edit Applications List** button, the Application Configuration Tool opens, which you can use to add and configure your applications for testing. If the application information has already been entered, you can edit the information. You also use the Application Configuration Tool dialog box to add applications. For information on editing or adding application information, see the Application Configuration Tool on page 1735.

Tasks view

The Tasks View displays errors, warnings, or other information automatically generated by the compiler.

By default, this view lists all tasks for all files in the project. You can apply a filter here by clicking the **Filter** button  in the Tasks View banner. You may want to restrict the Tasks View to showing tasks associated with the current script.

Refer to the online *Workbench User Guide* for more information.

To open: In the Functional Test Perspective, click the **Tasks** tab.

Test object maps

The Rational® Functional TesterHCL OneTest™ UI test object map lists the test objects in the application-under-test. It is a static view that describes test objects known to Rational® Functional TesterHCL OneTest™ UI in the application-under-test.

A test object map can include objects from multiple applications. The test object map provides a quick way to add test objects to a script. Because the test object map contains recognition properties for each object, you can update the information in one location. Any scripts that reference that test object map share the updated information.

When you record a script, Rational® Functional TesterHCL OneTest™ UI creates a test object map (or uses an existing shared map) for the application-under-test. Each script is associated with a test object map file. The map file can be private, associated with only the script (*.rftxmap), or shared between scripts (*.rftmap).

You can merge multiple private or shared test object maps into a single shared test object map.

You can create a new test object map to customize the information associated with a script. You can edit an existing test object map, add objects to the map, delete objects, update the recognition properties or unify two objects in the map. Additionally, you can delete a test object map.

You can insert dynamic objects. A test objects in the object map are in a hierarchy. The hierarchy of the objects may change if new objects are introduced in the test application. This results in a playback failure. Using dynamic test objects you can anchor a test object as a descendant to its parent making the playback resilient to hierarchy changes. Anchoring a test object to its parent will ensure that only direct descendants of the test objects are searched.

You can display a list of scripts associated with a test object map. You can use this list to add test objects to multiple scripts.

Test object inspector

You use the Test Object Inspector to view properties of selected test objects visible in the running application and display information about those objects, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information.

The Test Object Inspector displays information for the test object under the cursor.

Test object inspector menu

The test object map menu has the following options:

File Menu Options:

Exit -- Closes the Test Object Inspector window.

Option Menu Options:

Hide not mappable -- Parent test object nodes that do not appear in the object map are not included.

Hexadecimal -- Displays in hexadecimal Long, Integer, Short, and Byte number properties in the properties page.

Show rectangles -- In the Parent Hierarchy view, displays screen coordinates for the test object. This option is not enabled as a default, because it may slightly slow display time.

Show domains -- List the domain name for each test object.

Show Standard Properties Only -- Displays only standard properties, which provide a common way to access properties and their values across platforms.

Always On Top -- When selected, the Test Object Inspector window is always the topmost window on the screen.

Show Interfaces -- Display interfaces implemented by the classes in the Inheritance hierarchy.

Show Full Names -- Show full class names. For example, show "java.awt.Button" rather than "Button".

Show Menu Options:

Show Parent Hierarchy -- Displays the directed list of parents as in the test object map. Parent test object nodes that will not appear in the object map are labeled **Not Mapped**.

Inheritance Hierarchy -- Displays the class inheritance hierarchy for the selected test object, including any interfaces supported at each level of the hierarchy.

Show Properties -- Displays the object properties that appear in an object properties verification point associated with the selected test object.

Show Nonvalue Properties -- Displays a set of complex properties such as buttons and check boxes, which are not treated as values in the script. You can access these properties indirectly.

Show Method Information -- Displays the visible methods for the selected test object. These are the methods and necessary information to use the methods using the method available for every test object.

Applications Menu Options:

Run -- Opens the Start Application dialog box on page 1837, which enables you to start a specific application and add test objects to the test object map.

The Applications Menu also lists all the applications that have been configured.

Help Menu Options:

Help -- Displays online Help for the Test Object Inspector.

Test Object Inspector Toolbar

The Test Object Inspector toolbar has the following buttons:

 **Show Parent Hierarchy** -- Displays the directed list of parents as in the test object map. Parent test object nodes that do not appear in the object map are labeled **Not Mapped**.

 **Show Inheritance Hierarchy** -- Displays the class inheritance hierarchy for the selected test object, including any interfaces supported at each level of the hierarchy.

 **Show Test Object Value Properties** -- Displays the object properties that appear in an object properties verification point associated with the selected test object.

 **Show Test Object Nonvalue Properties** -- Displays a set of complex properties such as buttons and check boxes, which are not treated as values in the script. You can access these properties indirectly.

 **Show Method Information** -- Displays the visible methods for the selected test object. These are the methods and necessary information to use the methods using the method available for every test object.

 **Pause** -- Temporarily suspends the tool.

 **Resume** -- Enables you select another object in the application.

To open: For Rational® Functional TesterHCL OneTest™ UI, Eclipse Integration, from the product menu, click **Run > Test Object Inspector**. For Rational® Functional TesterHCL OneTest™ UI, Microsoft Visual Studio .NET Integration, from the Functional Test menu, click **Tools > Test Object Inspector**. From the toolbar, click the **Open Test Object Inspector** button  .

- For an enabled Java™ or already-infested application, Test Object Inspector tracks the cursor and performs live updates immediately after you open the application.
- If the application is not active, Test Object Inspector does not capture objects in the application. You must pause on the application to force the infestation before Test Object Inspector can track the cursor and perform live updates against that application. To pause on the application, hover the mouse over the object in the application, and press Shift. Test Object Inspector captures the object.
- When you press Shift to select the test object, Test Object Inspector copies the information displayed in the Test Object Inspector window to the system Clipboard.

Unify Test Objects dialog box

You use the first page of the Object Map: Unify Test Objects wizard to combine two objects within an object map, unify their properties, and automatically fix scripts to refer to the combined object.

The Unify Test Objects dialog box has the following controls:

Total number of source test objects affected -- Displays the number of test objects associated with the selected source object that will also be unified.

Total number of target test objects affected -- Displays the number of test objects associated with the selected target object that will also be affected.

Unified Test Object Properties -- Lists the Administrative and Recognition properties the unified object will have.

Source: *object* -- Lists the Administrative and Recognition properties of the source test object. To include any of the source test object properties in the unified test object properties, double-click the source property or copy/paste into the **Unified Test Object Properties**.

Target: *object* -- Lists the Administrative and Recognition properties of the target test object. To include any of the target test object properties in the unified test object properties, double-click the target property or copy/paste into the **Unified Test Object Properties**.

Right-clicking a property in any of the three grids displays these various options:

Open -- Displays the value in a separate window, which enables you to see long lines of text.

Case Sensitive Regular Expression -- Toggles case-sensitive comparison on and off.

Evaluate Regular Expression -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you use it.

Convert Value to Regular Expression -- Converts the recognition property value to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

Undo/Redo Regular Expression -- Converts the regular expression back to the original value.

Convert Value to Numeric Range -- Converts the recognition property value in the grid to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

Undo Numeric Range -- Converts the numeric range back to the original value.

Cut -- Removes the selected line from the grid and puts it on the Clipboard.

Copy -- Places a copy of the selected item on the Clipboard.

Paste -- Inserts the contents of the Clipboard into the grid.

Delete -- Removes the selected line from the grid.

Next -- Displays the second page of the Unify Test Objects wizard on page 1842, which lists all the scripts that reference the test object and that will be affected.

Finish -- Deletes the source object(s) from the object map and automatically updates all scripts that reference the unified test object(s).

Cancel -- Closes the Unify Test Objects dialog box without replacing the object.

To open: In the test object map, select a source object and drag it over the target object.

Unify Test Objects dialog box -- page 2

You use the second page of the Unify Test Objects wizard to confirm that you want to replace one test object with another.

The second page of the Unify Test Objects wizard has the following controls:

Affected Script References -- Lists all the scripts that reference the source and target objects that will be affected by the replacement.

Back -- Returns to the first page of the Unify Test Objects wizard on page 1841, which lists the Administrative and Recognition properties for the source object on the left and for the target object on the right.

Finish -- Deletes the source object(s) from the object map and automatically updates all scripts that reference the unified test object(s).

Cancel -- Closes the Unify Test Objects dialog box without replacing the object.

To open: When replacing one object with another, click **Next** in the Unify Test Objects dialog box.

Update Test Object Recognition Properties wizard

You use the Update Test Object Recognition Properties wizard to update recognition properties for a test object in the software under test.

The Update Test Object Recognition Properties wizard has the following controls and information:

Updated Test Object Properties for *object* -- Lists the updated Administrative and Recognition properties the object will have. You can double-click properties in the **Original Recognition Properties** and the **All Active Properties** sections to include them in this grid. Icons indicate the origin of the property:

Indicates that the property was added from the **All Active Properties** list.

Indicates that the property was added from the original recognition properties.

Indicates that the property was added from the original recognition properties and the **All Active Properties** list.

 Indicates that the property was manually edited.

Apply -- Enables you to edit the property and apply the edit without closing the page

Reset -- Enables you to restore the original list of updated test object properties. Reset removes all edits and additions you have made to the **Updated Test Object Properties** list.

Original Recognition Properties -- Lists the Administrative and Recognition properties the object had prior to the update. You cannot edit these properties. You can double-click a property or right-click and copy and paste a property into the **Updated Test Object Properties** grid.

All Active Properties -- Lists all the Administrative properties and all properties actively available for the object. You cannot edit these properties. You can double-click a property or right-click and copy and paste a property into the **Updated Test Object Properties** grid.

Right-clicking in any of the three grids displays a menu with the following options:

- **Add to Unified Test Object Properties** -- Copies the selected line in either the **Original Recognition Properties** or the **All Active Properties** grid to the **Updated Test Object Properties** grid.
- **Open** -- Displays the value in a separate window, which enables you to see long lines of text.
-  **Case Sensitive Regular Expression** -- Toggles case-sensitive comparison on and off.
-  **Evaluate Regular Expression** -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.
-  **Convert Value to Regular Expression** -- Converts the recognition property value in the **Updated Test Object Properties** grid to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.
-  **Undo Regular Expression** -- Converts the regular expression back to the original value.
-  **Convert Value to Numeric Range** -- Converts the recognition property value in the **Updated Test Object Properties** grid to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.
-  **Undo Numeric Range** -- Converts the numeric range back to the original value.
- **Cut** -- Removes the selected line from the **Updated Test Object Properties** grid and puts it on the Clipboard.
- **Copy** -- Places a copy of the selected item on the Clipboard.
- **Paste** -- Inserts the contents of the Clipboard into the **Updated Test Object Properties** grid.
- **Delete** -- Removes the selected line from the **Updated Test Object Properties** grid.

Back -- Redisplays the Choose Test Object to Update page.

Finish -- Completes updating recognition properties.

Cancel -- Closes the page without updating any recognition properties.

To open: Select an object in the test object map, start the application, and on the Test Object Map toolbar click the **Test Object: Update Recognition Properties** button  .

Variable Name page of the Verification Point and Action wizard

The Variable Name page is the fourth page of the Verification Point and Action Wizard when you choose **Get a Specific Property Value** as your action. On the Variable Name page, you can name the variable that holds the property value, and choose whether to declare the variable in your script.

You can open the Verification Point and Action Wizard with the **Insert Verification Point or Action Command** button  on the **Recording** toolbar. You can use the wizard to select objects to test in your application, and to select the types of tests to perform on them. The Select an Object page on page 1827 is the first step. After you select an object and click **Next**, the Select an Action page on page 1820 appears. On the Select an Action page, you choose an action to perform on the test object. Two of the actions are verification points (Properties or Data) and two of the actions, get a specific property value and wait for an object, are scripted actions against the object. If you choose **Get a Specific Property Value**, Rational® Functional TesterHCL OneTest™ UI displays the **Insert getProperty Command** page. The **Variable Name** page will be the fourth page when you click **Next**.

This page contains the following fields:

Object - Displays the name of the object for which you are getting a property.

Property - Displays the single property you chose to get.

Data Type - Displays the data type you are testing. This depends on the specific property you chose.

Variable Name - Accept the default suggestion listed in this box, or type a new name. The default name is based on the name of the object and the property you chose to test. After you accept or edit the name, click **Finish**.

Declare the variable in the script - This is selected by default. You need to declare a variable the first time you use the variable name. If you use the same variable name again in the same script, clear this option after the initial instance.

For more information, see Getting a Property Value on page 885.

Verification Point and Action wizard

You open the **Verification Point and Action wizard** dialog box with the **Insert Verification Point or Action Command** button  on the **Recording** toolbar. You use the wizard to select objects to test in your application, and to select the types of tests to perform on them. You record a verification point by using this wizard.

The Verification Point and Action Wizard has the following pages. Click the page name below for information about how to use that page.

Select an Object on page 1827 -- where you select the object you want to perform a test on. There are three selection methods.

Select an Action on page 1820 -- where you select the test to perform on that object. You can create a Data verification point, create a Properties verification point, get a specific property value, or set a wait state for an object.

Variable Name on page 1844 -- this is the third page when you choose a single property value as your action, which you can use to name the variable, and to choose whether to declare the variable in your script.

Verification Point Data on page 1852 -- this is the last page when you choose a Data or Properties verification point as your action. In both cases, you can select the verification point data you want to include in the test.

Note: While recording, you can drag the **Insert Verification Point or Action Command** button off of the **Recording** toolbar to immediately start selecting an object in your application. This is a shortcut for selecting it from the Select an Object page of the Verification Point and Action Wizard. You will then be in the wizard after you select the object.

Select an Object page of the Verification Point and Action Wizard

The Verification Point and Action Wizard is opened with the **Insert Verification Point or Action Command** button on the **Recording** toolbar or the product toolbar. It is used to select objects or images to test in your application, and to select the types of tests to perform on them.

The **Select an Object** page is the first step. You use one of the selection methods on this page to select the object in your application you want to perform a test on. When the object is selected, its recognition properties are listed in the grid at the bottom of the page.

To perform an image verification test, use the **Capture Screen Image** tool to select the image or use the **Object Finder** tool to select the object and create an image verification point.

If you select the wrong object, or decide to test a different one, use any of the methods and select a different object. It will then be shown in the grid. Once you advance past this first page by clicking the **Next** button, the object that is listed in the grid is the one that you will be testing. After you select a test object, you'll select an action in the next page of the wizard. Once you have moved on, you can always click the **Back** button to select a different object.

The following three selection methods are available:

Object Finder Tool

Use this tool to select an object and all descendents of the object, select one object, or select an object and the immediate children of an object.

This is the most common and direct method of selecting an object. Grab the Object Finder tool icon with your mouse and the cursor turns into the tool. Drag it over the object in your application that you want to select. You'll see that the object is highlighted and the object name is displayed. When you release the mouse button, the object is selected, and its recognition properties are listed in the grid.

Note that you can also use the **Insert Verification Point or Action Command button** **on the Recording** toolbar directly to select an object. If you click it and drag it off of the toolbar, it will become the object selector tool from this page of the wizard.

If the **After selecting an object advance to next page** option is selected, you'll go directly to the next page of the wizard after you select the object. Clear this option if you want to remain on this page to see the object recognition properties after selecting the object.

Object Browser

Use this method to browse for the object that you want to select. The browser displays a hierarchical tree of objects in your application. The top level shows any applications you have running. Under each top level,Rational® Functional TesterHCL OneTest™ UI displays the object hierarchy within that application. It is a dynamic view of the currently available objects.

Using this method, you browse for your object. The browser displays a hierarchical tree of objects in your system that are testable. The top level shows any applications you have running, and under each one is the object hierarchy within that application. It is a dynamic view of the currently available objects. Browse the object tree till you find the object, then click it. That will select it, and its recognition properties will be listed in the grid.

Delay Method

Use this method to select pop-up objects, such as menus. This method uses the Object Finder tool, but enables you to set a delay, which gives you time to get to an object that requires clicking on other objects first.

This uses the Object Finder tool, but with a delay that you set. The delay gives you time to get to an object that requires clicking on other objects first, such as a menu command. Set the number of seconds (the default is 10), then click the tool icon. Move your mouse to hover over your application until you get to the object you want to select. Anything you do during that delay period is not recorded. This allows you to "dig" for objects if necessary. For example, you might click with your mouse to cause a menu to open. The timer counts down, and when it runs out the object under the cursor is selected, and its recognition properties will be listed in the grid.



Note: In order for the delayed location of objects to play back correctly, the object must be exposed by actions in the script. If the object is not exposed, an Object Not Found Exception is thrown during playback.

If the **After selecting an object advance to next page** option is selected, you'll go directly to the next page of the wizard after you select the object. Clear this option if you want to remain on this page to see the object recognition properties after selecting the object.

Object recognition properties grid

When the object is selected by any of the above methods, its recognition properties are listed in the grid at the bottom of the page. The recognition properties are determined by the object's proxy. For example, a "button" object has three recognition properties: label, .class, and .classIndex. This grid will list the name and value of your specific object's recognition properties. You can use that information to confirm that you selected the correct object. If no information is listed, the object is not testable or the environment may not be enabled.

Capture Screen Image

To perform image verification test, use the **Capture Screen Image**  tool to capture the screen. This tool captures the full image of the screen.

Your next step

After you select an object using one of the methods listed above, click the **Next** button to choose an action to perform on the object. These include creating a data verification point, creating a properties verification point, creating an image verification point, getting a single property value, or setting a wait state on an object. For more information on the actions, see the Select an Action Page on page 1820.

Select an Action page of the Verification Point and Action wizard

You open the Verification Point and Action Wizard with the **Insert Verification Point or Action Command** button on the **Recording** toolbar. Use the wizard to select objects to test in your application, and to select the types of tests to perform on them. This is how you record a verification point.

The Select an Object page on page 1827 is the first step. After you select an object, the **Select an Action** page appears.

Note: By default, the **After selecting an object advance to next page** check box is checked. If you clear the option, you must click **Next** to advance to the next page after you select an object.

On the Select an Action page, you choose an action to perform on the test object. The Select an Action page has the following four actions:

Perform Data Verification Point -- Insert a verification point Data command on page 1784 for the selected object to test the data in your object when you play back your script.

Perform Properties Verification Point -- Insert a Properties verification point command on page 1783 for the selected object to test the properties in your object when you play back your script.

Perform Image Verification Point -- Insert an Image verification point command on page 1849 for the selected object to test the image in your object when you play back your script.

Get a Specific Property Value -- Insert a getProperty command on page 1782 for the selected object to put a getProperty into your script and return the value during playback.

Wait for Selected Test Object -- Insert a waitForExistence command on page 1784 to set a wait state for an object during playback to check for its existence.

Select an Action page of the Verification Point and Action wizard (from Insert)

When you open the Verification Point and Action Wizard while inserting a verification point from the Script Explorer, the **Select an Object** page appears.

Use the wizard to select objects to test in your application, and to select the types of tests to perform on them.

Here is how you create a verification point. The Select an Object page on page 1827 is the first step. After you select an object, the **Select an Action** page appears.

Note: By default, the **After selecting an object advance to next page** check box is checked. If you clear the check box, you must click **Next** to advance to the next page after you select an object.

When inserting a verification point while not recording, the Select an Action page is where you choose which verification point to perform on the test object. The Select an Action page has the following verification points:

Perform Data Verification Point -- Insert a verification point Data command on page 1784 for the selected object to test the data in your object when you play back your script.

Perform Properties Verification Point -- Insert a Properties verification point command on page 1783 for the selected object to test the properties in your object when you play back your script.

Perform Image Verification Point -- Insert Image verification point command on page 1849 for the selected object to compare the image when you play back your script.

Insert Verification Point Data Command page

Use to create a Data verification point for the selected object. The Data verification point tests the data in your object when you play back your script. The object name is listed at the top of the page. The list of tests shown in the **Data Value** field depends on information provided by the object proxy. Select the data value that you want to test.

Under **Verification Point Name**, accept the suggested default, or type a new name in the box.

Use the **Include Retry Parameters** to set a retry time for a verification point during playback to check for its existence. The retry option is useful when playback does not find the verification point in your application. To set a retry time, either use the default or set your own time. **Maximum Retry Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI tries again for the verification point to be visible in your application during playback. **Retry Interval** is the number of seconds that Rational® Functional TesterHCL OneTest™ UI checks for the verification point during the wait period.

When you select **Include Retry Parameters**, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the verification point in your application every 2 seconds, for up to 20 seconds. To set your own time, clear the default fields and type in your own values for **Maximum Retry Time** and **Retry Interval**. When you click **Finish**, the retry for verification point is written into your script, and occurs on future playbacks.



Note: When you insert a Data verification point without recording, the Include Retry Parameters option does not appear on the Insert Verification Point Data Command page.

To proceed with the verification point, click **Next**. For more information, see Creating a Data Verification Point on page 907.

Note: If your object has no data, this page is disabled.

Insert Image Verification Point Command page

Use this page to create an image verification point for the selected object or the image. The image verification point compares the image of the selected object or the specified image region with the object in the application under test when you play back the script.

The Insert Image Verification Point Command page has the following controls:

Verification Point Name

Accept the suggested default name or type a new name.

Select full image

To create an image verification point for the selected object or the full screen captured using Capture Screen Image tool.

Select a region of the image

To capture a region of the selected object or the screen to perform image verification test. Use the Region Tool icon to select the region.

Select region button

This option is enabled if you want to select a region of the image. Select the Region Tool icon and drag it over the image to position the mouse pointer at the starting point of the region. Without releasing the left mouse button, right-click and drag it to highlight the region of the image.

Selected region's dimension

This option is enabled if you choose to select a region of the image. The **X** and **Y** co-ordinates from the upper-left corner of the screen for the selected region are displayed. The total **Width** and **Height** of the selected region are also displayed. If required, you can edit the values instead of recapturing the region of the image.

Include Retry Parameters

Select this check box to set a retry time for a verification point during playback. Retry time provides time for the object to be displayed before the test checks for its existence. The retry option is useful when playback does not find the verification point immediately in your application. Use the default values or set your own time. The following controls are enabled:

- **Maximum Retry Time**: Accept the default or type the maximum number of seconds. Rational® Functional TesterHCL OneTest™ UI retries up to the specified time for the verification point to appear in your application during playback.
- **Retry Interval**: Accept the default or type the number of seconds. Rational® Functional TesterHCL OneTest™ UI checks the application every specified interval for the verification point to be displayed.

Note: When you insert an image verification point without recording, the Include Retry Parameters option is not displayed on the Insert Image Verification Point Command page

Insert Properties Verification Point Command page

Use to create a Properties verification point for the selected object. The Properties verification point tests the properties in your object when you play back your script. The object name is listed on the page. This verification point tests all properties of the object. You can edit the properties list later if you want to test only some of the properties.

Use the **Include Children** field to specify whether to include the properties of any child objects. **None** tests the object only (no children), **Immediate** tests the object and any immediate children (one level down), and **All** tests the object plus all of its children down the entire hierarchy.

Under **Verification Point Name**, accept the suggested default, or type a new name in the box.

Use the **Use standard properties** option to specify whether to use standard property types. If you are testing Java™, all properties are standard. Clear the option only if you are testing HTML and want to test browser-specific properties.

Use the **Include Retry Parameters** to set a retry time for a verification point during playback to check for its existence. The retry option is useful when playback does not find the verification point in your application. To set a retry time, either use the default or set your own time. **Maximum Retry Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI tries again for the verification point to be shown in your application during playback. **Retry Interval** is the number of seconds between times that Rational® Functional TesterHCL OneTest™ UI will check for the verification point during the wait period.

When you select **Include Retry Parameters**, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the verification point in your application every 2 seconds, for up to 20 seconds. To set your own time, clear the default fields and provide your own values for **Maximum Retry Time** and **Retry Interval**. When you click **Finish**, the retry for verification point is written into your script and occurs on future playbacks.

Note: When you insert a Properties verification point without recording, the Include Retry Parameters option does not appear on the Insert Properties Verification Point Command page.

To proceed with the verification point, click **Next**. For more information, see Creating a Properties Verification Point on page 904.

Note: If your object has no properties, this page is disabled.

Insert getProperty Command page

Use to get a single property value for the selected object. Functional Test puts a getProperty command into your script and returns the value during playback. This information is useful when you need to make a decision based on the property. For example, you might want to query whether a button was enabled.

When you select an object, the property list is built and displayed in the **Property Name** and **Value** fields. Select the property that you want to get. Click the **Next** button to proceed. The getProperty command is written into your script at the point you inserted it.

Note: If your object has no properties, this page is disabled.

Insert waitForExistence Command page

Use to set a wait state for an object during playback to check for its existence. The waitForExistence command is useful when waiting for an object right after your application starts, or after other actions that may take a long time.

The selected object is listed at the top of the page. To set a wait state for it, either use the default or set your own time. **Maximum Wait Time** is the maximum number of seconds Rational® Functional TesterHCL OneTest™ UI waits for the object to appear in your application during playback. **Check Interval** is the number of seconds between times that Rational® Functional TesterHCL OneTest™ UI checks for the object during the wait period.

When **Use the Defaults** is selected, Rational® Functional TesterHCL OneTest™ UI checks for the existence of the object in your application every 2 seconds, for up to 120 seconds. To set your own time, clear the default field and provide your own values for **Maximum Wait Time** and **Check Interval**. When you click **Finish**, the wait-for object is written into your script, and will occur on future playbacks. For more information, see Setting a Wait State for an Object on page 886.

Variable Name page of the Verification Point and Action wizard

The Variable Name page is the fourth page of the Verification Point and Action Wizard when you choose **Get a Specific Property Value** as your action. On the Variable Name page, you can name the variable that holds the property value, and choose whether to declare the variable in your script.

You can open the Verification Point and Action Wizard with the **Insert Verification Point or Action Command** button on the **Recording** toolbar. You can use the wizard to select objects to test in your application, and to select the types of tests to perform on them. The Select an Object page on page 1827 is the first step. After you select an object and click **Next**, the Select an Action page on page 1820 appears. On the Select an Action page, you choose an action to perform on the test object. Two of the actions are verification points (Properties or Data) and two of the actions, get a specific property value and wait for an object, are scripted actions against the object. If you choose **Get a Specific Property Value**, Rational® Functional TesterHCL OneTest™ UI displays the **Insert getProperty Command** page. The **Variable Name** page will be the fourth page when you click **Next**.

This page contains the following fields:

Object - Displays the name of the object for which you are getting a property.

Property - Displays the single property you chose to get.

Data Type - Displays the data type you are testing. This depends on the specific property you chose.

Variable Name - Accept the default suggestion listed in this box, or type a new name. The default name is based on the name of the object and the property you chose to test. After you accept or edit the name, click **Finish**.

Declare the variable in the script - This is selected by default. You need to declare a variable the first time you use the variable name. If you use the same variable name again in the same script, clear this option after the initial instance.

For more information, see Getting a Property Value on page 885.

Verification Point Data page of the Verification Point and Action wizard

The Verification Point Data page is the last page when you choose a Data, Properties or Image verification point as your action. In the Verification Point Data page, you can select the verification point data you want to include in the test.

You can open the Verification Point and Action Wizard with the **Insert Verification Point or Action Command** button  on the **Recording** toolbar. In the wizard, you select objects or the screen to test in your application, and select the types of tests to perform on them. The Select an Object page on page 1827 is the first step. After you select an object and click **Next**, the Select an Action page on page 1820 appears. This is where you choose an action to perform on the test object. If you choose **Perform Properties Verification Point**, **Perform Data Verification Point** or **Perform Image Verification Point**, the **Verification Point Data** page will be the final page.

After you view or edit the data in this page, click **Finish** to finish recording the verification point.

Metadata

The metadata is displayed in the left pane of the window. It displays a set of properties that define how specific data is managed. This grid can be edited. For example, you could edit the 'ignore case' or 'white space rule' in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

Properties grid display -- for a properties verification point

The object properties display in a grid format in the right pane. The properties in the grid belong to the object that is highlighted in the **Test Objects** tree. The properties appear in the left column and their values appear in the right column. You can edit which properties get tested in the **Property** column, and you can edit the property values themselves in the **Value** column.

By default, all properties will appear with no check mark, which means they will not be tested. Choose which properties you want to test by checking each of them. Checked properties will be tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button above the grid. Use the **Uncheck All** button to clear all properties. Depending on how many properties you want to test, it is often easiest to either select or clear all of them using one of those buttons, and then individually select or clear exceptions. It's a good idea to only test the specific properties you are interested in when you use a Properties verification point.

The grid uses a nested tree hierarchy. If a folder shows up on the list, you can expand it by double-clicking on it or selecting the expand (+) icon. If you select or clear the folder icon itself, all the properties underneath it will be tested or not tested.

To edit a value, double-click the grid cell. That cell becomes editable. Click outside the cell to make the edit take effect. In most cases, double-clicking a value makes the cell an editable field, and you can change only the value. In some special cases, another dialog box comes up containing the information. For example, if the property is color, when you double-click the color value, the standard Color dialog box opens. Make your edit there and close the Color box. In other cases, a drop-down list may appear in the **Value** column when you double-click a value. For example, values that are either true or false will appear in a drop-down list.

The grid has the following toolbar buttons for the Properties verification point display. These buttons act only on the currently displayed data.

Cut -- Cuts the selected property. It is placed on the Editor clipboard and can be pasted.

Copy -- Copies the selected property to the Editor clipboard.

Paste -- Pastes the cut or copied property from the Editor clipboard.

Delete -- Deletes the selected property. It will not be retained on the Editor clipboard.

 **Case Sensitive Regular Expression** -- Toggles case-sensitive regular expression comparison on and off.

 **Convert Value to Regular Expression** -- Converts the property value to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

 **Convert Value to Numeric Range** -- Converts the property value to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

 **Evaluate Regular Expression** -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.

 **Convert Value to dataset Reference/**  **Undo dataset Reference** -- Uses a dataset reference on page 951 to use a dataset instead of a literal value in a verification point. Cancels the dataset reference in the verification point on page 951.

 **Check All** -- Puts a checkmark in front of every property in the list. Checked properties will be tested each time you play back the script with this verification point.

 **Uncheck All** -- Clears the checkmark in front of every property in the list. Cleared properties will not be tested when you play back the script with this verification point.

 **Hide the Unchecked Properties//Show All Properties** -- Click **Hide the Unchecked Properties** to hide the cleared properties. Then you will only see the properties that will be tested. Click **Show All Properties** to display all properties, including any cleared ones.

Data display -- for a data verification point

You can display Data verification points in five ways, depending on which test you do on the data. The interface that appears in this dialog box is the same as what appears in the Verification Point Editor after you record the verification point.

For information on each of the five possible data displays, see the Verification Point Editor on page 917, and go to one of the following sections in that topic:

Data Verification Point--Menu Hierarchy Display

When you create a Data verification point and choose the Menu Hierarchy or Menu Hierarchy with Properties test, the menus display in a tree format. Menu Hierarchy and Menu Hierarchy with Properties are examples. The list of tests in the **Data Values** field is dependent on information provided by the object's proxy. Values other than these two might appear in the list.

Data Verification Point--Text Display

When you create a Data verification point and choose the Visible Text test, the text displays in a text box format. Visible Text is one example. The list of tests in the **Data Values** field is dependent on information provided by the object's proxy. Values other than this one might appear in the list.

Data Verification Point--Table Display

When you create a Data verification point and choose the Table Contents or Selected Table Cells test, the table data displays in a table format. Table Contents and Selected Table Cells are examples. The list of tests in the **Data Values** field is dependent on information provided by the object's proxy. Values other than these might appear in the list.

Data Verification Point--Tree Hierarchy Display

When you create a Data verification point and choose the Tree Hierarchy test, the data displays in a tree format. Tree Hierarchy is one example. The list of tests in the **Data Values** field is dependent on information provided by the object's proxy. Values other than this one might appear in the list.

Data Verification Point--List Display

When you create a Data verification point and choose the List Elements test, the data displays in a list format. List Elements is one example. The list of tests in the **Data Values** field is dependent on information provided by the object's proxy. Values other than this one might appear in the list.

Image display -- for an image and OCR verification point

The captured image is taken as the image verification point and is displayed in the right pane. Scroll bars are visible if the image size does not fit in the right pane of the page. For OCR, verification point, the characters of the selected image is also displayed below the displayed image pane.

Test object data in the Verification Point Data page

While inserting the verification points, if you have not checked the **Record Test Object relative Verification Points** option available in the General Recorder page on page 843 of the **Windows > Preferences** window, you can view the following test object data in the Verification Point Data page:

- Test objects
- Recognition and Administrative data

Test objects

The Test Objects pane is the upper left pane of the **Verification Point Data** page. The Test Objects tree is a partial version of the script's object map. This hierarchical display includes only the objects in your verification point. You cannot edit the Test Objects tree. You can choose an object within it and edit its properties in the properties list in the right pane.

You can double-click folders in the tree to expand and collapse the objects beneath them. Click an individual object in the tree to see its properties in the properties list.

Recognition and Administrative data

The recognition data is in the lower left pane. The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI and is not editable. The **Administrative** tab displays internal administrative data of the object. The Recognition and Administrative properties manage and describe the test object. Recognition and administrative data are the properties from the script's object map that locate and manage the test object in the context of the associated script. You can use this information to figure out what test object this is in the associated application under test.

The **MetaData** tab displays a set of properties that define how specific data is managed. You can edit the Metadata grid . For example, you could edit the 'ignore case' or 'white space rule' in a text verification point in the Metadata grid. To edit, double-click the value in the **Value** column.

The Recognition and Administrative properties that display in the pane will become a snapshot of the object map properties for the test object at the time the verification point is created. The properties become historical information as the application evolves.

Verification point comparator

You can use comparators to compare verification point data after you play back a script with a verification point and to update the baseline file. If the verification point fails, the comparator shows both expected and actual values so that you can analyze the differences. You can then load the baseline file and edit or update it with the values from the actual file.

Opening the verification point comparator

Click **View Results** in the Rational® Functional TesterHCL OneTest™ UI HTML log to open the comparator.

Notes:

- If you encounter an error regarding the Java™ plugin when trying to launch the comparator from **View Results** in the HTML log, you must configure your plugin properly.
- You must use Microsoft Internet Explorer to open **View Results** as browsers such as Mozilla Firefox and Google Chrome are not supported.
- With automatically enabled test environments, you cannot open the verification point comparator by clicking **View Results** in the functional test HTML log. In such cases, open the corresponding project log file from the functional test project log in the Functional Test Projects view.

Using verification point comparator for functional test scripts played back from Rational® Quality Manager

If you play back the script from Rational® Quality Manager, and click **View Results** in the detailed log to open the comparator, you are prompted to log in to Rational® Quality Manager. This occurs when you use the **Load baseline to edit** or the **Replace baseline with actual value** functions.

Note: To open the comparator from the Rational® Quality Manager detailed playback log, ensure that Rational® Functional TesterHCL OneTest™ UI is installed on the workstation where you are opening the log. Additionally, ensure that the Next-Gen plugin is disabled on the workstation.

When you open the comparator, the **Log In to Rational Quality Manager** dialog box is displayed and the Rational® Quality Manager server name and project area are displayed. You must specify a valid user name and password to log in to Rational® Quality Manager.

The **Log In to Rational Quality Manager** dialog box is displayed only the first time you use **Load baseline to edit** or the **Replace baseline with actual value** functions during an active Windows session. You are not prompted to log in a second time unless you have started a new Windows session and logged on to Windows as a different user.

Comparing verification points after playback

If you have one failed verification point, and you are using a log, select the log in the Functional Test Projects view. Right-click the log, and click **Failed Verification Points**. If you have one failed verification point, and you are using a log, select the log in the Solution Explorer. Right-click the log, and click **Failed Verification Points**. The verification point comparator is displayed.

If you have multiple failed verification points, and you are using a log, the **Results for Verification Points** wizard is displayed. Click a failed verification point in the list and click **View Results** or **Finish**. The comparator banner displays the name of your verification point.

You can specify color settings for several elements in the verification point comparator.

To edit verification point data, you must load the baseline by clicking the **Load Baseline to Edit** toolbar button  .

The verification point comparator window

The following sections describe the various components of the verification point comparator window and the toolbars.

Metadata

Metadata is displayed in the left pane of the verification point comparator window. It displays a set of properties that define how specific data is managed. You can edit this grid. For example, you could edit the *"ignore case "*or *"white space rule"* in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

Main toolbar

The toolbar at the top of the verification point comparator has six buttons.

 **File: Save**: Saves any changes you have made.

 **File: Revert**: Reverts to the state of the data at the last save you made. If you have not saved any changes since opening the comparator, it reverts to the state when it was opened. If you have edited and saved the changes, it reverts to the state at your last Save.

 **Load Baseline to Edit**: Loads the baseline file so you can edit it. The baseline values are displayed instead of the expected values. These values can be edited individually or replaced with actual values.

 **Replace Baseline with actual value**: Replaces the baseline values with all the values in the actual file. Then those values will become the baseline for future playbacks. If you want to replace only some of the values, edit them individually. This command replaces the entire file.

 **Hide/Show TestObject Info**: Toggles the display of the **Test Objects** and **Recognition Data** panes of the comparator window. When this information is hidden, the entire comparator window is used for the main data area. This is a sticky setting and is displayed as you last set it when you open the comparator later.

 **Note:** If your test objects tree has multiple nodes, the verification point comparator displays these panes again the next time you open it, regardless of the **Hide/Show TestObject Info** setting.

 **Help**: Displays the help documentation for the verification point comparator. You can open the Rational® Functional TesterHCL OneTest™ UI help any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Menu bar

The menu bar contains the same commands that are represented with the toolbar buttons described in this topic.

File: Displays the commands **Save**, **Revert**, **Baseline**, and **Replace**.

Edit: Displays the commands **Check All**, **Uncheck All**, and **Hide**. This menu is grayed out until you load the baseline for editing (using the **Load Baseline to edit** toolbar button ).

Difference: Displays the commands **First**, **Previous**, **Next**, and **Last**.

Test Object > Highlight: You can use this menu item if you need to verify an object in the application. If your test application is open, you can select an object in the test objects tree and then click this command to see the object highlighted in the application.

Preferences : Toolbars: You can use this menu item to control the display of the toolbars.

- **Test Object Appearance on the Tree**: Displays the **Edit Test Object Description** dialog box, which allows you to customize the text displayed for each object in the Test Object Hierarchy.
- **Hide TestObject Info**: You can use this command to toggle the display of the **Test Objects** and **Recognition Data** panes of the comparator window.

Help: Displays help documentation for verification point comparator.

Main data area

The right pane of the verification point comparator displays the verification point data. For example, in the case of a properties verification point, the **Property** and **Value** columns are displayed here. You can compare the verification point data here. If the verification point fails when you play back the script, the expected and actual values are displayed, irrespective of the type data display being used. In certain cases, the expected values are shown on the left pane and the actual values are shown on the right pane of the verification comparator window. In other cases, the values are displayed contiguously (such as nodes in a tree view), and the expected and actual values are shown in different colors if they are different. The expected value is red and the actual value is green in color. The actual values those that were recorded when you playback the script.

You can get seven types of displays from recording verification points, as described in the following sections, after the next section, Navigation Toolbar Buttons.

Navigation toolbar buttons

These four navigation buttons jump to the differences between the expected and actual files or the baseline and actual files. Differences are shown in red. The currently selected difference is highlighted.

 **Jump to First Difference**: Goes to the first difference in the expected/baseline and actual files.

 **Backward to Previous Difference**: Goes backward to the previous difference in the expected/baseline and actual files.

 **Forward to Next Difference**: Goes forward to the next difference in the expected/baseline and actual files.

 **Jump to Last Difference**: Goes to the last difference in the expected/baseline and actual files.

You can get the following types of displays after recording a verification point.

Properties verification point : grid display

When you create a properties verification point, the object properties are displayed in a grid format. The properties displayed on the grid belong to the object that is highlighted in the **Test Objects** tree. The properties appear are displayed in the left column and their values appear are displayed in the right column of the object properties grid.. You can edit which properties get tested in the **Property** column, and can edit the property values themselves in the **Value** column.

Properties with no check mark are not tested. You can select which properties you want to test by checking each of them. The checked properties are tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button  . You can use the **Uncheck All** button  to clear all properties. Depending on how many properties you want to test, it is often easiest to either select or clear all of them using one of those buttons, and then individually select or clear exceptions.

The grid uses a nested tree hierarchy. If a folder shows up on the list, you can expand it by double-clicking it or selecting the expand icon. If you check or clear the folder icon itself, all the properties underneath are either tested or not tested.

To edit a value, you must double-click the grid cell. Click outside the cell to make the edit take effect. In most cases, double-clicking a value makes the cell an editable field, and you can just change the value. In some special cases, another dialog box is displayed which contains the information. For example, if the property is color, when you double-click the color value, the standard color dialog box is displayed. You must your edit there and close the color box. In other cases, a drop-down list might be displayed in the **Value** column when you double-click a value. For example, values that are either true or false are displayed in a drop-down list.

The grid has the following toolbar buttons for the properties verification point display. In the comparator, these buttons are displayed only when you are editing the baseline.

 **Check All**: Includes a check mark in front of every property in the list. Checked properties are tested each time you play back the script with this verification point. Only checked properties are compared in the Comparator.

 **Uncheck All**: Clears the check mark in front of every property in the list. Do not test the cleared properties when you play back the script with this verification point.

 **Hide the Unchecked Properties/Show All Properties**: Click **Hide the Unchecked Properties** to hide the cleared properties. Then you only view the properties that are tested. Click **Show All Properties** to display all properties, including any cleared ones.

The grid has the following pop-up menu commands for the properties verification point display. To access them, right-click a value in the **Value** column.

Open: Displays the value in a separate window if the value is a string or a complex value type which enables you to see long lines of text and makes it easier to edit.

 **Case Sensitive Regular Expression**: Toggles case-sensitive regular expression comparison on and off.

 **Evaluate Regular Expression**: Displays the regular expression evaluator, which enables you to test the regular expression before you use it in a verification point.

 **Convert Value to Regular Expression**: Converts the property value to a regular expression.

 **Undo/Redo Regular Expression**: Cancels or redoes the regular expression conversion.

 **Convert Value to Numeric Range**: Converts the property value to a numeric range.

 **Undo Numeric Range**: Cancels the numeric range.

 **Convert Value to dataset Reference**: Uses a dataset reference to use a dataset instead of a literal value in a verification point.

 **Undo dataset Reference**: Cancels the dataset reference in the verification point.

 **Replace Baseline On Current Selection**: Replaces the baseline value with the actual value for just the selected property. This is a per-property version of the **Replace Baseline With Actual Value** toolbar button  .

Compare object properties

To compare object properties, look at the expected or baseline values and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which are displayed in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : menu hierarchy display

When you create a data verification point and choose the menu hierarchy or menu hierarchy with the properties test, the menus are displayed in a tree format in the main data area. Menu hierarchy and menu hierarchy with properties are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be displayed.

The tree displays the entire menu hierarchy of your application, or one top-level menu and its subitems, depending on how you recorded the verification point. If you chose the whole menu bar, each top-level menu is displayed in the tree, in the same order they are displayed in the menu bar. Each individual menu item is displayed under its top-level menu. You can use the plus and minus signs to open and close the list for each top level menu.

To edit a menu, double-click it in the tree. You must load the baseline first before doing this. The menu properties displayed in a grid, which you can then edit. You can edit the actual values by double-clicking a value in the **Value** column. You can also edit the list of properties that are tested during playback by using the check box beside each property. The checked items are tested. The toolbar buttons above the grid are the same ones that are found in the

object properties grid, except for **Hide/Show**. The buttons work the same, except they apply to the selected menu property or value.

Compare menu hierarchy data

To compare menu hierarchy data, look at any differences shown in red and green. The expected values are displayed in red, and the actual values are shown underneath them in green. The actual values are what were captured when you played back the script. If the descriptions for the expected and baseline values are the same, but if there are some differences in their properties, the node is displayed as blue in color. You can use the navigation buttons to navigate to all the differences. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : text display

When you create a data verification point and choose the Visible Text test, the text is displayed in a text box format in the main data area. For example, visible text. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be displayed.

The text is displayed in a text box area. You cannot edit directly in this area. To edit the verification point data, click **Edit Text**  above the data display. You must load the baseline file before doing this. A small text editor containing the text is displayed. You can edit the text in this editor, and when you close it, the edited text is displayed in the baseline column of the comparator.

Compare text data

To compare text data, look at the expected and actual values columns. The actual values are those that were captured when you play back the script. You can use the navigation buttons to navigate to all the differences, which is displayed in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : table display

When you create a data verification point and choose the table contents or selected table cells test, the table data is displayed in a table in the main data area. Table Contents and selected table cells are two examples. The list of tests is displayed in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these may also appear.

The table displays the same information as the table in your application. To edit the verification point data, double-click any cell in the table to edit that cell. You must load the baseline file before doing this.

You can also edit which cells in the table get tested. Table cells that are within the comparison regions are shown with a grey background. If you are testing the entire table, all cells will be grey. You can use the drop-down list in the toolbar above the data region as a selection mechanism. (This doesn't show up until you load the baseline.) Choose **Column**, **Row**, or **Cell Selection** in the list, then make your selections in the table. For example, if you select **Row Selection**, when you click a cell in the second row, the whole second row will be selected. If you had chosen **Cell Selection**, only that cell would have been selected. After you select the data you want to compare, click the **Update Comparison Region** button  to have your changes take effect.

The **Cut**, **Copy**, **Paste**, and **Delete** toolbar buttons above the table area apply to the selected row(s), and are only applicable within the Verification Point Comparator. (It does not use the system clipboard.)

You can right-click a table item to access a pop-up menu. The commands are the same as those listed above in the **Properties Verification Point--Grid Display** section.

There are features in the **Metadata** tab that you can also use to edit the table data. For example, you can edit the table's column headers or row headers by accessing them in the **MetaData** tab. To edit column headers, double-click the **Value** column of the **columnHeaders** property. A small editor opens that lets you edit the headers. The row headers work the same way if your table has them. Double-click the **rowHeaders Value** to edit them. In order for the column headers to be compared, you must change the **compareColumnHeaders** property to **true** in the MetaData tab. The **compareRowHeaders** value works the same way to indicate whether row headers will be compared.

If you double-click the **Value** of the **compareRegions** property in the Metadata tab, an editor will open showing the selected regions of your table. For selected sells, it shows the row index or key value pairs and the column header or index of each selected cell. For selected rows, it shows the row index or key value pairs. For selected columns, it shows the column header or index. Using this compare regions editor is another way you can select which regions get compared. If you click the **Compare All Cells** button  in this editor, all of the table cells will be tested.

If your table supports row keys or column keys, you can edit those and insert keys by double-clicking on the **columnKeys** and **rowKeys** values in the **Metadata** tab.

Compare table data

To compare table data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which appear in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point -- tree hierarchy display

When you create a Data verification point and choose the Tree Hierarchy test, the data is displayed in a tree format in the main data area. For example, Tree Hierarchy. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this might be displayed.

The tree displays the entire tree hierarchy in your application or the part of the tree selected when you create the verification point. Each item in the tree is displayed in the same order it is displayed in your application. Each individual item is displayed under its top-level item. You can use the plus and minus signs to open and close the list for each top-level item.

To edit an item in the hierarchy, double-click it in the tree. A small text box is displayed, which you can use to edit the item.

Compare tree hierarchy data

To compare tree hierarchy data, look at any differences displayed in red and green. The expected values are displayed in red, and the actual values are displayed in green. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences.

Data verification point : list display

When you create a data verification point and choose the List Elements test, the data is displayed in a list format in the main data area. List Elements is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this might be also displayed.

The list displays the same information as the list in your application, and in the same order. To edit a list item, double-click it in the list display. (If you have not done so, you must load the baseline first.) You can also edit the list of which items get tested during playback by using the check box beside each item. The checked items are tested.

The toolbar buttons preceding the list are the same ones that are found in the object properties grid described above in the Properties Verification Point : Grid Display section. The buttons work the same as described there, except they apply to the selected list item(s).

You can right-click a table item to access a pop-up menu. The commands are the same as those listed preceding the **Properties Verification Point : Grid Display** section.

Compare list data

To compare list data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can use the navigation buttons to navigate to all the differences, which are shown in red. You can edit the baseline values or replace the baseline with the actual file.

Data verification point : state display

When you create a data verification point and choose the Check Box Button State or Toggle Button State test, the data is displayed in a list format in the main data area. Check Box Button State or Toggle Button State are two examples. The list of tests displayed in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this may be also displayed.

Compare state data

To compare state data, look at the expected and actual values columns. The actual values are those that were captured when you played back the script. You can edit the baseline values or replace the baseline with the actual file.

Test object data in the Verification point comparator window

While inserting the verification points, if you have not checked the **Record Test Object relative Verification Points** option available in the General Recorder page of the **Windows > Preferences** window, you can view the following test object data in the Verification Point comparator:

- Test objects
- Recognition and Administrative data

Test objects

This is the upper left pane of the Verification Point Comparator window. It is a partial version of the script's object map. This hierarchical display includes only the objects in your verification point. You cannot edit the Test Objects tree. You can choose an object within it and edit its properties or data in the right pane of the Verification Point Comparator window.

You can double-click folders in the tree to expand and collapse the objects beneath them. You must an individual object in the tree to see its properties or data in the right pane.

The check boxes to the left of each node Verification Point Comparator window indicate whether that node is tested or not. Checked items get tested. After you load the baseline to edit, you can select or clear items.



Note: If your test application is open, you can select an object in the Test Objects tree and then click **Test Object > Highlight** from the Verification Point Comparator menu to see the object highlighted in the application. You must use this feature if you need to verify an object in the application.

Recognition and Administrative data

This is the lower left pane of the Verification Comparator window. The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI and is not editable. Some of these properties are the recognition properties that were listed in the **Select an Object** tab of the Verification Point and Action Wizard when you created the verification point. The **Administrative** tab displays internal administrative data of the object and is not editable. These properties are used to manage and describe the test object. Recognition and administrative data are the properties from the script's object map used to locate and manage this test object in the context of the associated script. You can use this information to determine what test object this is in the associated application under test.

The **MetaData** tab displays a set of properties that define how specific data is managed. This grid can be edited if you load the baseline. For example, you could edit the *"ignore case"* or *"white space rule"* in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

The Recognition and Administrative properties are a snapshot of the object map properties for the test object at the time the verification point was created. They become historical information as the application evolves.

Related reference

Edit Test Object Appearance dialog box on page 1757

Related information

Comparing and updating verification point data using the Comparator on page 933

Enabling the Java plug-in of a browser on page 800

Viewing logs in the Projects view on page 1501

Editing test object descriptions

Replacing an exact-match property with a pattern

Verification Point Editor

The Verification Point Editor lets you view and edit verification point data. You can open the Editor by double-clicking a verification point in the Script Explorer window. The Editor banner displays the name of your verification point.

You can specify color settings for several elements in the Verification Point Editor.

The following sections explain the parts of the Verification Point Editor window, and the toolbars.

Metadata

The metadata is displayed in the left pane of the window. It displays a set of properties that define how specific data is managed. This grid can be edited. For example, you can edit the 'ignore case' or 'white space rule' in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

Main toolbar

The toolbar at the top of the Verification Point Editor has five buttons.

 **File: Save** -- Saves any edits you have made.

 **File: Revert** -- Reverts to the state of the data at the last save you made. If you have not saved edits since opening the verification point, it will revert to the state it was in when opened. If you have done editing and made saves, it will revert to the state at your last Save.

 **Hide/Show TestObject Info** -- Toggles the display of the **Test Objects** and **Recognition Data** panes of the Editor window. When this information is hidden, the entire Editor window is used for the main data area. This is a sticky setting--the next time you open the Editor it will appear as you last set it. However, note that if your Test Objects tree has multiple nodes, the Verification Point Editor will show these panes again the next time you open it, regardless of this setting.

 **Replace Baseline** -- Replaces the baseline image with a new image. The new image will become the baseline for future playbacks. The Verification Point and Action Wizard is invoked for recapturing the image verification point.

 **Help** -- Brings up the Help for the Verification Point Editor. You can open the Rational® Functional TesterHCL OneTest™ UI help at any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Menu bar

The menu bar contains the same commands that are represented with the toolbar buttons described in this topic.

File -- These are the same **Save**, **Revert**, **Check Out**, and **Exit** commands as the buttons listed above in the Main Toolbar section.

Edit -- These are the same commands as the buttons listed below in the Properties Verification Point section.

Test Object > Highlight -- If your test application is open, you can select an object in the Test Objects tree and then click this command to see the object highlighted in the application. Use this feature if you need to verify an object in the application.

Preferences > Toolbars -- Toolbars controls the display of the toolbars. Hides/Displays the File, Metadata and Help toolbars. **Test Object Appearance on the Tree** displays the Edit Test Object Description dialog box on page 1757, which enables you to customize the text displayed for each object in the Test Object Hierarchy. **Hide TestObject Info** toggles the display of the **Test Objects** and **Recognition Data** panes of the Editor window.

Help -- Displays the Help for the Verification Point Editor. You can open the Rational® Functional TesterHCL OneTest™ UI Help any time from the **Help** menu in Rational® Functional TesterHCL OneTest™ UI.

Main data area

The right pane of the Verification Point Editor is where the verification point data is displayed. For example, in the case of a Properties verification point, the **Property** and **Value** columns are displayed here. This is where you edit the verification point data.

There are seven different types of displays you can get from recording verification points, as described in the following sections.

Properties Verification Point -- Grid Display

When you create a Properties verification point, the object properties are displayed in a grid format. See Creating a Properties Verification Point on page 904 for information on recording it. The properties that are shown in the grid belong to the object that is highlighted in the **Test Objects** tree. The properties appear in the left column and their values appear in the right column. You can edit which properties get tested in the **Property** column by checking a check box for a property, and can edit the property values themselves in the **Value** column.

By default, all properties will appear with no checkmark, which means they will not be tested. Choose which properties you want to test by checking each of them. Checked properties will be tested each time you play back a script with this verification point. You can check all properties in the list by clicking the **Check All** toolbar button  above the grid. Use the **Uncheck All** button  above the grid to clear all properties. Depending on how many properties you want to test, it is often easiest to either select or clear all of them using one of those buttons, and then individually select or clear exceptions. It's a good idea to just test the specific properties you are interested in when you use a Properties verification point.

The grid uses a nested tree hierarchy. If a folder shows up on the list, you can expand it by double-clicking on it or selecting the expand icon. If you select or clear the folder icon itself, all the properties underneath it will be tested or not tested.

To edit a value, double-click the grid cell. That cell will then be editable. Click outside the cell to make the edit take effect. In most cases double-clicking a value makes the cell an editable field, and you can just change the value. In some special cases, another dialog box comes up containing the information. For example, if the property is color,

when you double-click the color value, the standard Color dialog box opens. Make your edit there and close the Color box. In other cases, a drop-down list may appear in the **Value** column when you double-click a value. For example, values that are either true or false will appear in a drop-down list. If the value is a string or a complex value type, you can right-click the value and select **Open** to display the value in a separate window, which enables you to see long lines of text and makes it easier to edit.

Note: You can change a property value to a regular expression or numeric range using the Verification Point Editor. For information, see Replacing an Exact-Match Property with a Pattern.

The grid has the following toolbar buttons for the Properties verification point display. These buttons act only on the currently displayed data.

Cut -- Cuts the selected property. It is placed on the Editor clipboard and can be pasted.

Copy -- Copies the selected property to the Editor clipboard.

Paste -- Pastes the cut or copied property. It will be inserted into the display in alphabetical order.

Delete -- Deletes the selected property. It will not be retained on the clipboard.

Case Sensitive Regular Expression -- Toggles case-sensitive comparison on and off.

Convert Value to Regular Expression -- Converts the recognition property value in the **Updated Test Object Properties** grid to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

Convert Value to Numeric Range -- Converts the recognition property value in the **Updated Test Object Properties** grid to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

Evaluate Regular Expression -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.

Convert Value to dataset Reference/ **Undo dataset Reference** -- Uses a dataset reference to use a dataset instead of a literal value in a verification point. Cancels the dataset reference in the verification point. See About dataset References and Verification Points on page 951.

Check All -- Puts a checkmark in front of every property in the list. Checked properties will be tested each time you play back the script with this verification point.

Uncheck All -- Clears the checkmark in front of every property in the list. Cleared properties will not be tested when you play back the script with this verification point.

Hide the Unchecked Properties/Show All Properties -- Click **Hide the Unchecked Properties** to hide the cleared properties. Then you will only see the properties that will be tested. Click **Show All Properties** to display all properties, including any cleared ones.

The grid has the following pop-up menu commands for the Properties verification point display. To access them, right-click a value in the **Value** column.

Open -- If the value is a string or a complex value type, this will display the value in a separate window, which enables you to see long lines of text and makes it easier to edit.

 **Case Sensitive Regular Expression** -- Toggles case-sensitive regular expression comparison on and off.

 **Evaluate Regular Expression** -- Displays the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a verification point.

 **Convert Value to Regular Expression** -- Converts the property value to a regular expression. See Replacing an Exact-Match Property with a Pattern for more information.

 **Redo/Undo Regular Expression** -- Redoes or cancels the regular expression conversion.

 **Convert Value to Numeric Range** -- Converts the property value to a numeric range. See Replacing an Exact-Match Property with a Pattern for more information.

 **Undo Numeric Range** -- Redoes or cancels the numeric range.

 **Convert Value to dataset Reference** - - Uses a dataset reference on page 951 to use a dataset instead of a literal value in a verification point.

 **Undo dataset Reference** -- Cancels the dataset reference in the verification point on page 951.

Data Verification Point--Menu Hierarchy Display

When you create a Data verification point and choose the Menu Hierarchy or Menu Hierarchy with Properties test, the menus are displayed in a tree format in the main data area (right pane). Menu Hierarchy and Menu Hierarchy with Properties are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be shown.

The tree will display the entire menu hierarchy of your application, or one top-level menu and its sub-items, depending on how you recorded the verification point. If you chose the whole menu bar, each top-level menu will be shown from top to bottom in the tree in the order they appear from left to right in the menu bar. Each individual menu item is shown under its top-level menu. Use the plus and minus signs to open and close the list for each top-level menu.

By default, all menu items will appear with a check mark, which means they will be tested. Checked items will be tested each time you play back a script with this verification point, and cleared items will not be tested. You can check all menu items by clicking the **Check All** toolbar button above the tree. Use the **Uncheck All** button to clear all items.

The **Cut**, **Copy**, **Paste**, **Delete** , **Check All**, and **Uncheck All** toolbar buttons above the tree apply to the selected menu item in the tree hierarchy, and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

Data Verification Point--Text Display

When you create a Data verification point and choose the Visible Text test, the text is displayed in a text box format in the main data area (right pane). Visible Text is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

The text is displayed in a text box that can be used like a very basic text editor. You can type and edit directly in this text box. To edit the verification point data, make your edits to the text in this area.

Data Verification Point--Table Display

When you create a Data verification point and choose the Table Contents or Selected Table Cells test, the table data is displayed in a table in the main data area (right pane). Table Contents and Selected Table Cells are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these may be shown.

The table displays the same information as the table in your application. To edit the verification point data, double-click any cell in the table to edit that cell.

The **Cut**, **Copy**, **Paste**, and **Delete** toolbar buttons above the table area apply to the selected row(s), and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

You can right-click a table item to access a pop-up menu. The commands are the same as those listed above in the **Properties Verification Point--Grid Display** section.

Data Verification Point--Tree Hierarchy Display

When you do a Data verification point and choose the Tree Hierarchy or Selected Tree Hierarchy test, the data is displayed in a tree format in the main data area (right pane). Tree Hierarchy and Selected Tree Hierarchy are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than these two may be shown.

The **Cut**, **Copy**, **Paste**, **Delete** , **Check All**, and **Uncheck All** toolbar buttons above the tree apply to the selected item in the tree hierarchy, and are only applicable within the Verification Point Editor. (It does not use the system clipboard.)

Data Verification Point--List Display

When you create a Data verification point and choose the List Elements test, the data is displayed in a list format in the main data area (right pane). List Elements is one example. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

The toolbar buttons above the list are the same ones that are found in the object properties grid described above in the Properties Verification Point--Grid Display section. The buttons work the same as described there, except they apply to the selected list item(s). The **Cut**, **Copy**, **Paste**, and **Delete**, **Check All**, and **Uncheck All** toolbar buttons are only applicable within the Verification Point Editor. (It does not use the system clipboard.) The **Insert** button is described above.

Data Verification Point--State Display

When you create a Data verification point and choose the CheckBox Button State or Toggle Button State test, the data is displayed in a list format in the main data area (right pane). CheckBox Button State or Toggle Button State are two examples. The list of tests shown in the **Data Value** field is dependent on information provided by the object's proxy. Values other than this one may be shown.

Test object data in the Verification point editor window

While inserting the verification points, if you have not checked the **Record Test Object relative Verification Points** option available in the General Recorder page on page 843 of the **Windows > Preferences** window, you can view the following test object data in the Verification Point editor:

- Test objects
- Recognition and Administrative data

Test objects

This is the upper left pane of the Verification Point Editor window. It's a partial version of the script's object map. This hierarchical display includes only the objects in your verification point. You cannot edit the Test Objects tree. For a Properties verification point, you can choose an object within it and edit its properties in the properties list in the right pane.

You can double-click folders in the tree to expand and collapse the objects beneath them. Click an individual object in the tree to see its properties in the properties list.

The check boxes to the left of each node indicate whether that node will be tested or not. Checked items get tested.

Note: If your test application is open, you can select an object in the Test Objects tree and then click **Test Object > Highlight** or right-click an object and click **Highlight** from the Verification Point Editor menu to see the object highlighted in the application. Use this feature if you need to verify an object in the application.

Recognition and Administrative data

This is the lower left pane of the Editor window. The **Recognition** tab displays recognition data used by Rational® Functional TesterHCL OneTest™ UI and is not editable. The **Administrative** tab displays internal administrative data of the object and is not editable. These properties are used to manage and describe the test object. Recognition and administrative data are the properties from the script's object map used to locate and manage this test object in the context of the associated script. You can use this information to figure out what test object this is in the associated application under test.

The **MetaData** tab displays a set of properties that define how specific data is managed. This grid can be edited. For example, you could edit the 'ignore case' or 'white space rule' in a text verification point in this metadata grid. To edit, double-click the value in the **Value** column.

The Recognition and Administrative properties are a snapshot of the object map properties for the test object at the time the verification point was created. They become historical information as the application evolves.

Web browsers tab of the Enable Environments dialog box

This dialog is opened by clicking **Configure > Enable Environments for Testing** from Rational® Functional TesterHCL OneTest™ UI. The **Web Browsers** tab is used to enable your browsers and to add and configure browsers. Information about enabling browsers is presented first. Information about adding and configuring browsers is presented below that.

For enabling web browsers:

The Rational® Functional TesterHCL OneTest™ UI HTML enabler is the **Web Browsers** tab of the Enable Environments dialog box. The HTML enabler must be run before you can use Rational® Functional TesterHCL OneTest™ UI to test HTML applications. It enables HTML applications running in that browser to be tested by Rational® Functional TesterHCL OneTest™ UI. On Windows® systems, the enabler looks in the registry to discover any installed browsers. On UNIX®, the enabler scans your hard disk drive(s) looking for any installed browsers.

The first time you run Rational® Functional TesterHCL OneTest™ UI, it automatically enables Internet Explorer. If you have Mozilla Firefox, you must enable them using the **Enable** button. If you install a new browser and want to use that browser for testing, you must rerun the HTML enabler after you complete the installation of the browser. You can run the enabler any time from Rational® Functional TesterHCL OneTest™ UI by clicking **Configure> Enable Environments for Testing**. See Enabling Web Browsers on page 780.

Note that the first time you run Rational® Functional TesterHCL OneTest™ UI it automatically enables the JVM of your browser's Java™ plug-in so that HTML recording works properly. If you install a different JVM, you must rerun the enabler to enable it.

Web browsers list

Displays the list of browsers that the enabler locates on your hard disk drive(s). This list is populated when the enabler starts up. On Microsoft® Windows® platforms, the enabler uses the registry to locate browsers. Browsers are identified by the full path name to their installation directory. After the name, the enabler indicates in parentheses whether that browser is currently enabled.

Select All button

Use this to select all the browsers that are listed in the **Web Browsers** list. This is useful if you want to enable or disable all the browsers. To clear them all, click any of the individual browsers.

Search button

Click this button to have Rational® Functional TesterHCL OneTest™ UI search your hard disk drive(s) for web browsers. This opens the Search for Web Browsers dialog box. Choose one of the search options in that dialog and click the **Search** button. **Note:** You should not use the **Search All** option to find browsers on Linux® or UNIX® systems. Instead use the **Search In** option to locate the browser. See Enabling Web Browsers on page 780 for

information on the search options. When the search is complete, the **Web Browsers** list is populated with all found browsers. At least the first time you use Rational® Functional TesterHCL OneTest™ UI, use the **Search** button to locate all browsers on your system. After the initial search, it will list any browsers that were already enabled, plus any new ones it finds.

Add button

Click this button to locate browsers individually. It brings up the Add Browser dialog box to locate the browser. To select a browser, point to its installation directory. The browser you select will be added to the **Web Browsers** list. The main use of **Add** would be if the enabler failed to locate a browser automatically at start-up. You can also use the **Search In** option in the Search for Web Browsers on page 1819 dialog box to locate a browser.

Remove button

If you want to remove a browser from the **Web Browsers** list, select it and click **Remove**.

Set as Default button

Use this to choose which browser you want to be your default browser. Select the browser in the list, and click the button. That browser will then become the default, and will be indicated in parentheses after the name. You can change the default any time by coming back to this tab.

Enable button

Use this button to enable selected browser(s) for testing with Rational® Functional TesterHCL OneTest™ UI. Select the browser(s) to enable in the list, then click **Enable**. The modifications to the browser to enable it are done at this time. Once enabled, that will be indicated in parentheses after each browser's name in the list.

Disable button

Use this button to disable selected browser(s) for testing with Rational® Functional TesterHCL OneTest™ UI. Select the browser(s) to disable in the list, then click **Disable**. This undoes all the modifications made by **Enable**, and the enabled indicator will disappear after the name.

Test button

You can test that your browser is enabled properly by clicking the **Test** button. This opens the Browser Enablement Diagnostic Tool on page 799. If you suspect your browser is not enabled properly, run this tool and follow the instructions it gives to solve the problem.

Note:

- To enable JREs for Java™ testing, click the Java™ Environments tab of the enabler and click the Help button, or see Enabling Java Environments on page 778.
- If your browser is not enabled, you will be able to tell because the Record Monitor on page 1807 will be blank when you try to record against an HTML application. For this reason, leave the record monitor in view while recording. If you see this symptom, you need to run the enabler.

For adding and configuring web browsers:

The **Web Browsers** tab is also used to add and edit browser configurations. To edit the information on an existing browser, click the name of the browser in the **Web Browsers** list. To add a new browser, click the **Search** or **Add** button. Use the **Set as Default** button to set one of the browsers as your default browser. Whether editing or adding, make your changes, then click **OK** for the changes to be saved.

Web browsers list

Select the browser that you want to edit or view. Its information will then appear to the right of the list. The information fields are described below. If your browser is not in the list yet, click **Search** or **Add** to find and enter it.

The browser that has default listed after it in parentheses is the default browser. It will be used in all HTML testing unless you change this setting in the properties of a specific application.

Detailed Information for *Browser*

Contains the following fields:

Name -- This is the logical name of your browser. It may be used in a script by a startBrowser command or by an HTML application in the Application Configuration Tool on page 1735. Rational® Functional TesterHCL OneTest™ UI defaults this from the end of your path. You can edit this name.

Kind -- This is a read-only field. Rational® Functional TesterHCL OneTest™ UI will fill it in based on which browser you select.

Path -- This is the full path to the root of the browser installation.

Command -- This is the browser's executable name. For example: "mozilla" or "iexplore."

Search button

Click **Search** to add all your browsers into the **Web Browsers** list. This opens the Search for Web Browsers dialog box. Choose one of the search options in that dialog and click the **Search** button. **Note:** You should not use the **Search All** option to find browsers on Linux® or UNIX® systems. Instead use the **Search In** option to locate the browser. See Enabling Web Browsers on page 780 for information on the search options. Rational® Functional TesterHCL OneTest™ UI will enter all the detailed information on each browser.

Add button

Click **Add** to manually locate a new browser to add to the list. The Add Browser dialog appears. Browse to the executable file of the browser you want to add. With the file selected, click the **Add** button. The browser will then show up in the list and you can edit its configuration information if necessary. Note: it may be quicker to use the **Search** button and let Rational® Functional TesterHCL OneTest™ UI find and enter your browsers.

Set as Default button

Use this to choose which browser you want to be your default browser when Rational® Functional TesterHCL OneTest™ UI starts up HTML applications. Select the browser in the list, and click the button. That browser will then become the default, and will be indicated in parentheses after the name. You can change the default any time by coming back to this tab.

Remove button

To remove a browser from the **Web Browsers** list, select it, then click **Remove**.

OK button

You must click **OK** when you are finished to save the additions or edits you made on this tab.

For more information, see Configuring Browsers for Testing on page 798.

Apply button

If you want to apply edits you make in this dialog box before you exit the dialog, click **Apply** . If you click **Cancel**, any changes you made before you clicked **Apply** will be saved, and changes made after will be canceled.

Workbench Preferences page

The Workbench Preferences page enables you to indicate how you want the Workbench to behave while playing back, recording, and debugging Functional Test scripts.

The Workbench page has the following controls:

Workbench state during run -- Enables you to indicate how you want the Workbench to display while playing back scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during playback.
- **Minimized and restored on playback termination (Default)** -- Reduces the Workbench to a button on the taskbar during playback and restores it when playback finishes.
- **Hidden** -- Hides the Workbench during playback and restores it when playback finishes.
- **Leave in current state** -- Does not change the Workbench during playback.

Workbench state during recording -- Enables you to indicate how you want the Workbench to display while recording scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during recording.
- **Minimized and restored when recording finished (Default)** -- Reduces the Workbench to a button on the taskbar during recording and restores it after recording stops.
- **Hidden** -- Hides the Workbench during recording and restores it after recording stops.
- **Leave in current state** -- Does not change the Workbench during recording.

Workbench state during debug -- Enables you to indicate how you want the Workbench to display while debugging scripts.

- **Minimized** -- Reduces the Workbench to a button on the taskbar during debugging.
- **Minimized and restored on playback termination** -- Reduces the Workbench to a button on the taskbar during debugging and restores it after debugging stops.
- **Hidden** -- Hides the Workbench during debugging and restores it after debugging stops.
- **Leave in current state (Default)** -- Does not change the Workbench during debugging.

Restore Defaults -- Restores all the default values on this page.

Apply -- Saves the edits you made without closing the dialog box.

To open: Click **Window > Preferences**. In the left pane expand **Functional Test** and click **Workbench**.

Workbench Advanced Preferences

This Advanced page enables you to set advanced Workbench preferences for IBM® Rational® Functional TesterHCL OneTest™ UI, such as switching to the Test Debug perspective rather than the Functional Test perspective when debugging or turning on or off.

The Advanced page has the following controls:

Switch to Test Debug Perspective when debugging -- When selected, IBM® Rational® Functional TesterHCL OneTest™ UI switches to the Test Debug perspective when you select **Script > Debug**. When cleared, IBM® Rational® Functional TesterHCL OneTest™ UI continues to display the Functional Test perspective while the script runs in debug mode.

Restore Defaults -- Restores all the default values on this page.

Apply -- Saves the edits you made without closing the Preferences dialog box.

To open: Click **Window > Preferences**. In the left pane expand **Product Name > Workbench**, and click **Advanced**.

Test Object Map menu

This topic describes each of the options on the test object map menu.

File menu on page 1876

Edit menu on page 1876

Find menu on page 1876

Test Object menu on page 1877

Preferences menu on page 1878

Applications on page 1878

Display on page 1878

Help on page 1878

File menu options

The File menu has the following commands:

Save -- Saves your changes to the test object map.

Revert -- Restores the map to the version last saved.

Renew All Names in Associated Script(s) -- Renews all the names of script test objects in associated script(s).

Exit -- Closes the test object map.

Edit menu options

Cut -- Removes properties selected on a property set tab to a local Clipboard.

Copy -- Copies properties selected on a property set tab to a local Clipboard.

Paste -- Inserts properties previously saved to a local Clipboard at the cursor location property set.

Delete -- Depending on which pane has focus, deletes the selected test object from the test object map or deletes properties from the property set.

Find menu options

Quick Find -- Opens the Quick Find dialog box, which enables you to search for a test object based on a string you specify.

Find by Filters -- Opens the Set Active Find Criteria dialog box on page 1835, which enables you to select the filter to use for searching the test object map or to create a new filter.

Find & Modify -- Opens the Find & Modify dialog box, which enables you to do either a Quick Find or a Find by filters and make modifications to the results.

Find Used -- Finds all the test objects that have references in the scripts associated with a shared test object map.

Find Not Used -- Finds test objects that have no references in scripts associated with the test object map.

Delete All Not Used -- Opens the Delete All Not Used Test Objects dialog box, which enables you to selectively delete test objects that do not have references in the script associated with the test object map.

First -- Moves to the first test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state. For information, see Searching for Objects in a Test Object Map.

Previous -- Moves to the previous test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state. For information, see Searching for Objects in a Test Object Map.

Next -- Moves to the next test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state. For information, see Searching for Objects in a Test Object Map.

Last -- Moves to the last test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state. For information, see Searching for Objects in a Test Object Map.

Test Object menu options

Insert Object(s) -- Opens the Insert a GUI Object into the Object Map dialog box, which enables you to select test objects to Adding objects to a test object map and make them available for scripts.

Update Recognition Properties -- Enables you to update the recognition properties of a test object frame in the application-under-test.

Description Property -- Opens the Set Description Property dialog box, which enables you to Adding test object descriptions. Rational® Functional TesterHCL OneTest™ UI adds the description to the Administrative property set tab for the object and displays the description when you place the cursor over the object name in a script.

Add to Script *script* -- Adding test objects to a script, which enables you to add it to the script and select a method. Rational® Functional TesterHCL OneTest™ UI changes this menu item to **Add to Multiple Scripts** to indicate that multiple scripts have been selected and are affected by the command.

Associated Scripts -- Displays a list of scripts that are associated with the test object map.

Accept Node -- Changes the state of the selected test object from **New** to "Clean."

Accept All -- Changes the state of all test objects from **New** to "Clean."

Highlight -- Locates the test object in the application-under-test, if visible. If Rational® Functional TesterHCL OneTest™ UI finds more than one instance of the test object, you can display neither or the top two candidates.

Renew Name in Associated Script(s) -- Renews the name of an individual script test object in associated script(s).

Preferences menu options

Toolbars -- Enables you to display or hide the File, Edit, Find, TestObject, Applications, Display, and Help toolbars.

Test Object Description -- Opens the Edit Test Object Description dialog box on page 1757, which enables you to customize the text displayed for each object in the Test Object Hierarchy.

Clear State on Close -- Accepts all test objects in the test object map by changing their state from **New** to "Clean" when you close the map.

Highlight -- Opens the Set Highlight Window Preferences dialog box on page 1836, which enables you to specify how to emphasize objects in the application-under-test when you select them.

Applications menu options

Run -- Opens the Start Application dialog box on page 1837, which enables you to start a specific application and add test objects to the test object map.

The Applications menu also lists up to nine of the most recently used applications. A number appears next to each application on the menu, which enables you to select an application by typing the number that corresponds to the application. You can also click the application name at the bottom of the Applications menu.

Display menu options

Expand All -- Displays all test objects in the hierarchy.

Collapse to Selected -- Closes all test objects in the hierarchy except in the selected tree.

Toggle Orientation -- Switches between displaying the property sets under or next to the Test Object Hierarchy.

Help menu options :

Test Object Map Help -- Displays Help for the test object map.

Insert Test Object Help -- Displays Help for adding objects to a test object map.

New Test Object Help -- Displays Help that describes how to set the option to display the test object map for a new object.

Test Object Map toolbar

This topic describes each button on the test object map toolbar

The Test Object Map toolbar has the following buttons:

 **File: Save** -- Saves changes you make to the test object map.

 **File: Revert** -- Restores the map to the version last saved.

 **Edit: Cut** -- Removes text selected in the property set tab to a local Clipboard.

 **Edit: Copy** -- Copies text selected in the property set tab to a local Clipboard.

 **Edit: Paste** -- Inserts text previously saved to a local Clipboard at the cursor location in the property set.

 **Edit: Delete** -- Deletes the selected test object from the test object map.

 **Find: Quick** -- Opens the Quick Find dialog box, which enables you to search for a test object based on a string you specify.

 **Find: Filters** -- Opens the Set Active Find Criteria dialog box, which enables you to select the filter you want to use for searching the test object map or to create a new filter.

 Find: Find & Modify -- Opens the Find & Modify dialog box, which enables you to use Quick Find or Find by filters and make modifications to the results.

 **Find: Used** -- Finds all the test objects that have references in the scripts associated with a shared test object map.

 **Find: Not Used** -- Finds test objects that do not have references in scripts associated with the test object map.

 **Delete All Not Used** -- Opens the Delete All Not Used Test Objects dialog box , which enables you to selectively delete test objects that do not have references in the script associated with the test object map.

 Find: First -- Moves to the first test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state.

 **Find: Previous** -- Moves to the previous test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state.

 **Find: Next** -- Moves to the next test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state.

 **Find: Last** -- Moves to the last test object in the hierarchy that matches the search criterion. The default criterion is State Not Clean, which searches the test object map for all objects that have a New state.

 **Test Object: Insert Object(s)** -- Opens the Insert a GUI Object into the Object Map dialog box, which enables you to select test objects to add to the test object map and make them available for scripts.

 **Test Object: Insert Dynamic Test Object** -- Opens the Add Dynamic Test Object in the Object Map dialog box. You can anchor a test object as a descendant to its parent.

 **Test Object: Source Object to Unify** -- Selects the source object. The source object is the old object that will be replaced by the new objects properties.

 **Test Object: Target Object to Unify** -- Unifies the source object with the target object. The older object is now replaced with the new object properties

 **Test Object: Update Recognition Properties** -- Enables you to update the recognition properties of a test object in the application-under-test.

 **Test Object: Description** -- Opens the Set Description Property dialog box, which enables you to enter descriptive text about the object. Rational® Functional TesterHCL OneTest™ UI adds the description to the Administrative Property Set tab for the object and displays the description when you place the cursor over the object name in a script.

 **Test Object: Add to Script: *script*** -- Adds the selected object to the Script Explorer, which enables you to add it to the script and select a method. Rational® Functional TesterHCL OneTest™ UI changes the text in the tooltip for this button to **Add to Multiple Scripts** to indicate that multiple scripts have been selected and will be affected by the command.

 **Test Object: Associated Scripts** -- Opens the Associated Scripts dialog box, which lists scripts associated with a test object map.

 **Test Object: Accept Node** -- Changes the state of the selected test object from **New** to "Clean."

 **Test Object: Accept All** -- Changes the state of all test objects from **New** to "Clean."

 **Test Object: Highlight** -- Locates the test object in the application-under-test, if visible. If Rational® Functional TesterHCL OneTest™ UI finds more than one instance of the test object, you can display neither or the top two candidates.

 **Test Object: Renew Name in Associated Script(s)** -- Renews the name of an individual script test object in associated script(s).

 **Application: Run** -- Opens the Start Application dialog box , which enables you to start an application and add test objects to the test object map.

 **Expand All** -- Displays all the test objects in the hierarchy.

 **Collapse to Selected** -- Closes all the test objects in the hierarchy except in the selected tree.

Display: Toggle Splitter Orientation -- Changes the property sets display from under the Test Object Hierarchy to beside it.

Help: Help -- Displays online Help for the test object map.

Test object hierarchy

The Test Object Hierarchy lists all test objects in the application-under-test and provides information for each, such as color, owner relationship, state, test domain, role, name, and .class.

- **Color** -- Newly added test objects are marked **New** and displayed in blue. All the test objects that are not used in the scripts associated with the test object map are displayed in red.
- **Owned** -- An owner/owned relationship is not a container relationship. For example, a frame and a dialog box. A parent/child relationship is a frame and a toolbar

  You can change the color of the Owned test object on page 830 in the test object map.

  You can change the color of the Owned test object in the test object map.
- **State**
  - **New** -- Added to the test object map from the Insert a GUI Object into the Object Map dialog box.
  - **"Clean"** -- The object has been accepted and any previous state has been cleared. A "clean" object is not labeled, and the state is removed from the hierarchy.
- Test domain
  - HTML
  - Java
  - Net
  - Win
- **Role** -- The generic type of an object, such as Frame or Button.
- **Name** -- The descriptive name administrative property.
- **.class**
  - Java class name, such as java.awt.Button
  - Html canonical class name, such as Html.HtmlDocument or Html.A

Each test object in the list is preceded by an icon that indicates its role.

Property sets

The lower (or right) pane of the Test Object Map window contains property sets, which provide information about the selected object.

There are two property set tabs:

- **Recognition**
- **Administrative**

The **Recognition** tab contains the recognition data used by Rational® Functional TesterHCL OneTest™ UI. The **Administrative** tab contains the internal administrative data of the object. These properties are used to manage and describe the test object. Updating the properties on this tab affects the future code generation of scripts that use this test object. For example, updating the Descriptive Name causes the new name to be used the next time this test object is added to a script, depending on the template used.

Recognition and administrative data are the properties from the object map that are used to locate and manage this test object in the context of the associated script. For information, see Using ScriptAssure (TM) on page 1441.

To edit a value in either of the tabs, double-click the value.

If you select a recognition property value and right-click, you can use any of these various options:

Open -- Displays the value in a separate window, which enables you to see long lines of text.

Case Sensitive Regular Expression -- Sets case-sensitive comparison in regular expressions on and off.

Evaluate Regular Expression -- Starts the Regular Expression Evaluator on page 1810, which enables you to test the regular expression before you try it in a test object find.

Convert Value to Regular Expression -- Converts the recognition property value in the **Updated Test Object Properties** grid to a regular expression. For more information, see Replacing an Exact-Match Property with a Pattern.

Undo Regular Expression -- Restores the original value of the regular expression.

Convert Value to Numeric Range -- Converts the recognition property value in the **Updated Test Object Properties** grid to a numeric range. For more information, see Replacing an Exact-Match Property with a Pattern.

Undo Numeric Range -- Converts the numeric range back to the original value.

You can change a property value to a regular expression or numeric range by using the test object map editor. For information, see Replacing an Exact-Match Property with a Pattern.

When a test object changes, you can update its recognition properties in the application-under-test.

You can Adding test object descriptions as a property on the Administrative tab for an object.

You can also specify color settings for several elements in a test object map.

Specify Playback Options page

Use the Specify Playback Options page to specify run arguments and a dataset iteration count.

The Specify Playback Options page has the following controls:

Run arguments -- Select to pass Command line interface on page 1726 to the script. The most recently used run arguments appear in the list.

dataset Iteration Count -- Determines how many times a test script runs when you play back the test script. Specify the count according to the number of records in the dataset. Type or select the number of records in the dataset, or select **Iterate Until Done** to access all records in the dataset. For a call script, you can select **Use Current Record** to use the same record across the call script.

To open:

In the Projects view, select a script and from theRational® Functional TesterHCL OneTest™ UI menu click **Script > Run**. Click **Next**. In the Solution Explorer, select a script and from the product menu, click **Script > Run**. Click **Next**.

Object Properties Configuration Tool

Use the object properties configuration tool to configure the object recognition properties in the customized object library. While recording scripts, the customized object library file is used as a reference for setting object recognition properties and the property weights in the object map.

The object properties configuration tool has the following controls:

Select the Test Domain

Lists the test domains for which you can configure the object recognition properties.

Select the Object Class

Lists all the default objects and the customized objects for the selected test domain.

Show only customized object classes

When selected, the Test Object Class field lists only the customized objects for the selected domain.

Add Object

To add an object if the required test object is not listed for the specific domain. You can also add the recognition property if you do not know the exact property name to an existing test object in the Add Test Object dialog box.

Remove Object

To remove a test object from the object library if it is not a default test object used by Rational® Functional TesterHCL OneTest™ UI. This button is disabled if you select a default test object.

Object Recognition Properties grid

Lists the recognition properties, actual weights and default weights of the selected object. The property weights are not displayed if multiple objects are selected in the Test Object Class field. The actual weights of the object property can be edited.

Import

To import objects along with its recognition properties and weights from an object properties file into the object library. The existing object in the object library will be overwritten if the existing object class name is same as the object that you are importing.

Export

To export objects along with its recognition properties and weights of the object library into an object recognition property file. The file is saved with .rftop extension.

Restore

To restore the actual property weights of the selected objects to the default weights.

Add

To add a recognition property for the selected object. An empty row is added in the Object Recognition Properties grid.

Remove

To remove a recognition property for the selected object.

Apply to selected objects

To apply a recognition property for multiple objects. Right-click the recognition property row that you want to apply to the selected objects and click Apply to selected objects.

Property Weight dialog box

This dialog box is displayed while applying a recognition property to multiple objects. Type the property weight in the dialog box and click Ok.

Remove from selected objects

To remove the recognition property from the selected objects. Right-click the recognition property row that you want to remove from the selected objects and click Remove from selected objects.

Finish

Saves the changes and closes the object properties configuration tool.

Cancel

Cancels all the changes made in the object properties configuration tool after the last save operation.

Apply

Saves the changes without closing the object recognition editor.

To open the object properties configuration tool, click **Configure > Configure Object Recognition Properties**

Add Object dialog box

Use this option to add new objects to the object library and specify properties that needs to be added as recognition properties to the test object.

The Add Object dialog box has the following controls:

Class name

Type the class name of the object to add it to the object library if you know the correct test object name.

Select properties from

Lists all the objects that are used in the object library. Select the object from the list to use the recognition properties and its weights in the new object.

Object Finder

Select the Object Finder tool icon and drag it over the object in the application that you want to select. Rational® Functional TesterHCL OneTest™ UI outlines the object with a highlight border.

Start Application

To start the test application for selecting the objects that you want to add to the object library.

Object Recognition Properties grid

Lists the object recognition properties and weights of the selected object. Select the check box corresponding to the property row to add the recognition property for the test object

OK

Saves the changes and closes the Add Object dialog box.

Cancel

Cancels all the changes made in the Add Object dialog box.

To open the Add Object dialog box, click **Configure > Configure Object Recognition Properties**. In the object properties configuration tool, click **Add Object** to open the Add Object dialog box.

Import Object Recognition Properties dialog box

Use this option to import objects along with its recognition properties and weights from an object properties file into the object library.

The Import Object Recognition Properties dialog box has the following controls:

Object Properties file

Displays the path of the selected object recognition properties file.

Browse

To browse and select the object recognition properties file from which you want to import the objects.

Select the objects to import

Lists the object details along with the domain names listed in the selected object recognition properties file. Select the check box corresponding to the objects that you want to import from the list.

Select all

To select all the check boxes corresponding to objects in the list.

Deselect all

To clear all the check boxes that is selected in the objects list.

Ok

Saves the changes and closes the Import Object Recognition Properties dialog box.

Cancel

Cancels all the changes made in the Import Object Recognition Properties dialog box.

To open the Import Object Recognition Properties dialog box, click **Configure > Configure Object Recognition Properties**. In the Object Properties Configuration Tool, click **Import** to open the Import Object Recognition Properties dialog box.

Export Object Recognition Properties dialog box

Use this option to export the customized objects along with its recognition properties and weights of the object library into an object recognition property file.

The Export Object Recognition Properties dialog box has the following controls:

Select the objects to export

Lists the customized object details along with the domain names of the object library. Select the check box corresponding to the objects that you want to export from the list.

Select all

To select all the objects displayed in the list.

Deselect all

To clear all the check boxes that is selected in the objects list.

Object properties file

Displays the path of the specified object recognition properties file.

Browse

To browse and specify the object recognition properties file to which you want to save the object details. The object recognition properties file is saved with .rftop extension.

Ok

Saves the changes and closes the Export Object Recognition Properties dialog box.

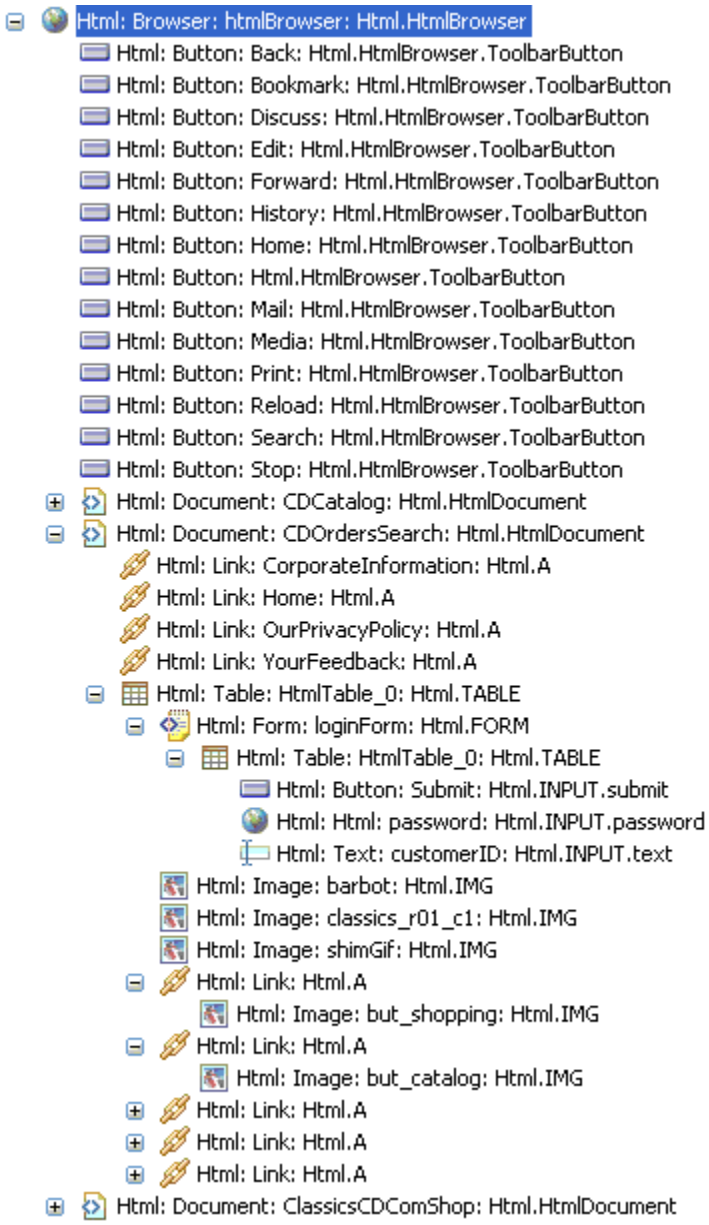
Cancel

Cancels all the changes made in the Export Object Recognition Properties dialog box.

To open the Export Object Recognition Properties dialog box, click **Configure > Configure Object Recognition Properties**. In the Object Properties Configuration Tool, click **Export** to open the Export Object Recognition Properties dialog box.

Example of a test object map

This is an example of a test object map created for an HTML application:



Exception dialog box

This dialog box is displayed if an exception occurs during playback of a functional test script.

The Exception dialog box has the following controls:

Description

Displays the reason for the exception.

Try fixing the problem

Select this check box to fix the problem. After you fix the problem, try the operation again. The following controls are enabled:

Start the application

Opens the **Start the application** dialog box to select and open the application-under-test.

Enable the environment

Opens the **Enable Environments** dialog box to enable the required environment for testing.

Find the object

Opens the **Find the object** wizard to select the required object in the application under test. The selected object is used to perform the action when you try the operation again.

Open the object map editor

Opens the object map editor to view and edit the object recognition properties. The updated object recognition properties are used when you try the operation again.

Configure this application

Opens the **Application Configuration Tool** to edit the application configuration. This option is available if the application is not configured.

Switch to manual mode

This option is displayed for the keyword-enabled functional test scripts that are executed from Manual Tester in hybrid mode. Using this option you can switch the execution from automated to the manual mode.

Retry the operation

Tries the execution of the script a second time.

Skip the operation

Continue to execute the next line of code. The playback log does not display the exception.

Stop execution

Stops the script playback.

Do not show this dialog again

Select this check box to disable the appearance of the exception dialog box while executing the script.

Application View

The **Application View** displays the application visuals (snapshots) that are captured while recording scripts. The visuals of the test application are captured only if both the simplified scripting and the application visuals feature preferences are enabled.

Rational® Functional TesterHCL OneTest™ UI captures the application controls and their data and property details during recording if the application visuals, data verification point and data driven commands features are enabled in

the Rational® Functional TesterHCL OneTest™ UI preferences window. With these features enabled, you can create or edit verification points in the script and insert data-driven commands from the application visuals that are displayed in the **Application View** without opening the test application.

When you click a simplified script test line, the application visual that contains the application control is highlighted in blue and is displayed in the **Application View**. The Thumbnails pane in the **Application View** displays the application visuals of all the test scripts in the project that are captured while creating the scripts.

You can modify the test script to test additional application controls, create or edit verification points, or insert data-driven commands by selecting the application controls in the **Application View** without opening the application under test.

The following controls are available in the **Application View** toolbar:

- **Hide/Show Comment**: The comments that are inserted for an application control in an application visual are displayed in the application view when you point the control.
- **Hide/Show all the visuals**: The thumbnails view displays all the application visuals in the project if you select Show all the visuals option. By default, the application visuals of the active script are displayed in the thumbnails view.
- **Control Highlight Color** and **Hover Highlight Color**. From the **View** menu, select the color from the palette for changing the control highlight color and the hover highlight color. By default, the control that is referred in the selected test line is highlighted in blue and hover highlight color is red.

Select any control in an application visual. The following right-click menu options for a control is available in the application view:

- **Insert *control_name* control**: To insert the control to the test script from the application visual. The action that can be performed on the selected control is also listed based on the type of the control in the application visual. A statement that specifies the control and the action is added as a test line into the test script.
- **Insert comment**: To add a comment for the control. The comment is displayed below the application visual.
- **Insert Verification Point**: Use this option to insert a test line to perform a data or an image verification for the control in the application under test during the script playback. This option is available only if the **Enable capturing of verification on test data** option is selected on the **Functional Test Preferences** window.
- **Insert Data Driven Commands**: To perform a data-driven test for the control by retrieving the input values for the control from the dataset during playback.
- **Update Visual**: Update the visual in the application view by selecting the visual from the application under test.

Script editor

The simplified script editor displays the test script as English statements that are easy to understand and edit.

All the recorded actions on the test application are displayed as test lines in the simplified script editor.

You can perform the following operations in the simplified script editor:

- **Edit the test line**: You can modify the input values of a test line. Click the test line and modify the input values.
- **Enable/Disable action**: Use this option to enable or disable the action on the application under test during playback.
- **Delete action** (): Use this option to delete the test line from the script editor.
- **Create Group** (): The test lines in the script editor are grouped based on the parent window that the test line control refers to. Use this option to create more logical groups to manage the test lines for easy identification.
- **Insert Java Code Snippet** () or **Insert Java Method** (): Use these controls to switch to Java test script.
- **Insert comments** (): Use this option to insert comments in the script editor.
- **Repeat Actions**(): Use this option to repeat the actions statements. The selected test lines are grouped into the `Repeat` group and are executed based on the repeat count during playback.
- **Insert Condition (If Clause)** (): Use this option to insert conditional statements to verify the values of the variables in the script and perform actions in the application. The test lines in the `If Then` group are executed if the conditions of the variables are met during playback.
- **Insert Else Clause** (): Use this option insert the `Else` group and add the test lines that must be executed if the variable conditions are not met during script playback.
- **Undo**: Use this option to undo the action that you performed in the simplified script editor.
- **Redo**: Use this option to redo the action that you performed in the simplified script editor.

The Java code for the simplified script is displayed in the Java script editor. The simplified script test line is displayed as a comment in the corresponding Java code.

The properties of each test line can be viewed and modified in the Properties View.

Properties view - General page

The **General** page of the **Properties** view displays the details of the test line that is selected in the script editor. You can change the test line description and the action on the control that the test line refers to. If a group is selected in the simplified script editor, this page shows the dataset details.

The **General**page of the **Properties** view contains these controls when a test line is selected in the script editor:

Control name

Displays the name of the control that the test line refers to. You can change the control name in this field.

Action

Lists the actions that can be performed on the control. Select an alternative action from the list to change the recorded action on the control, if required.

Action parameter

Displays the input values or the action details such as the screen coordinates or the path. You can change the values in this field.

Control type

Indicates the generic type of the control, such as a frame or button.

Application domain

Lists the domain to which the control belongs, such as the Java or Win domain.

Control state

Indicates the state of the control in the application such as the browser-ready state.

The **General**page of the **Properties** view contains the following dataset details when a group is selected in the script editor:

dataset name

Specify the dataset name to associate it with the group.

dataset iteration count

Specify how many times the statements in the group must be run during the script playback.

Remove dataset associated with the selected group

Select this check box to remove the dataset association with the selected group.

The **General**page of the **Properties** view contains the following conditional controls when an **If** or **Then** statement is selected in the script editor:

Left Side

Lists the variables that are declared during the script recording. This field is displayed if you select the **If** statement in the script editor.

Compares To

Lists the operator parameters. The following parameters are available:

- EQUALS
- CONTAINS
- STARTS_WITH
- ENDS_WITH
- LESS_THAN
- LESS_OR_EQUAL
- GREATER_THAN
- GREATER_OR_EQUAL

Right Side

Specify the variable value that must be verified. For string values, you must specify the value in quotation marks, for example, `"Visa"`. You can also select another variable from the list, if the **Left Side** value must be verified against another variable.

Properties View- Playback page

Use the **Playback** page of the **Properties View** to specify the playback settings for the test line to be run during the script playback.

The **Playback**page of the **Properties View** contains these controls:

Wait for the control to be displayed

Type the time in seconds to wait for the control to be displayed in the application under test before running the test line during script playback. Use this option when the application under test might take time to refresh the required screen or controls during playback.

Pause execution for

Type the time in seconds to delay running the test line during script playback.

Exception handlers

Specify the type of action that must be performed if an exception occurs while running the test line during script playback. You can select any action such as **Stop**, **Skip and Continue** , or **Retry** from the list for any of these exceptions:

- **Object not found**: This exception occurs if the control is not found in the test application during playback.
- **Ambiguous control**: This occurs if Rational® Functional TesterHCL OneTest™ UI cannot uniquely identify the control in the application under test. This typically happens when the application window is not closed before playback and also when an identical control is displayed in the same window.
- **Weak recognition**: This exception occurs if the control properties such as the control name or the properties of the parent control is changed in the application during playback.
- **Control item not found**: This exception occurs if the subitem control is not found in the application during playback.
- **Unexpected error**: Any other type of exception that is not listed earlier.

Properties View - Log page

Use the **Log** page of the **Properties View** to specify the information that must be displayed in the playback log when the selected test line is executed.

The **Log** page of the **Properties View** contains these controls:

Snapshot options

- **Control snapshot**
- **Screen snapshot**
- **None**

Select either the control snapshot or the screen snapshot option to specify the type of snapshot to be captured while running the test line during script playback.

Type

- **Information**
- **Warning**
- **Error**
- **None**

Indicate the type of message with the details that must be displayed in the log after the test line runs during the script playback.

Reference information for the Visual Studio IDE

In this section, you will learn all the reference or additional information about the Functional Test perspective in the Visual Studio IDE.

VB.NET scripting command-line interface

Some of the actions that can be performed from the Rational® Functional TesterHCL OneTest™ UI graphical user interface can also be performed from the VB.NET Scripting Edition command line.

These actions include:

- Compile a script
- Compile/re-compile an entire project
- Play back a script
- Add/remove Full Trust for a project to the .NET runtime security policy
- Write a .NET configuration for an executable or assembly to use an acceptable framework

Command line format

The core command line format to display the error messages in the command line window is:

rational_ft.exe <Rational® Functional TesterHCL OneTest™ UI VB.NET options>

The core command line format is:

rational_ftw.exe <Rational® Functional TesterHCL OneTest™ UI VB.NET options>

Note: The rational_ftw.exe file is similar to the rational_ft.exe file. However, the rational_ftw.exe file does not display the error messages in the console window.

Recording of scripts, environment enablement, and editing of assets are only available from the Rational® Functional TesterHCL OneTest™ UI Java™ command-line interface. For more information, see Command line interface on page 1726.

To record a VB.NET script using the Rational® Functional TesterHCL OneTest™ UI Java™ command-line interface, you can use the options described for recording with the language VB option. The language VB option specifies that the Java™ command-line recorder must generate the test scripts in the VB.NET language.

Common Rational® Functional TesterHCL OneTest™ UI VB.NET scripting command-line options

Some of the more common Rational® Functional TesterHCL OneTest™ UI VB.NET scripting command-line options are as follows:

Command-line option	Usage
-datastore <project directory>	Used to identify the project location.
-compile <script name>	Used to compile a specific script.
-compileall [<namespace>]	Used to incrementally compile the whole project namespace specified, or the entire project if no namespace is provided.
-recompileall [<namespace>]	Used to do a full compilation of the whole project namespace specified, or the entire project if no namespace is provided.
-playback <script name>	Used to play back a specific script.
-addfulltrust	Used to add Full Trust runtime security policy for the specified project (use datastore). With the .NET Framework, the default security policy prohibits playback of scripts from a project over the network.
-removefulltrust	Used to remove the Full Trust runtime security policy for the specified project (use datastore).
-writeconfig <path to application>	Used to have an application built with an older version of the .NET framework (.NET Framework 1.0 or above).

Exemple

Examples of command-line usage

Some examples of command-line usage are:

For more information, see Command line interface on page 1726.

Task	Command syntax
Record a new script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [**-language** vb **-map** <sharedmap>] [options] **-record** <script-name>
Record into an existing script, inserting before or after a given line	java **-classpath**<classpath> com.rational.test.ft.rational_ft **-datastore** <directory> [**-insertbefore** <line>] [**-insertafter** <line>] [options] **-language** vb **-map** <sharedmap>] [options] **-record** <script-name>
Compile a script **Note:** You must enable the Java environment on page 778 before doing this. You must also install Java SDK and add the bin directory to the path.	**rational_ft.exe -datastore** <directory> [options] **-language** vb **-compile** <script-name>
Play back a script, passing command-line arguments <values> to the script	**rational_ft.exe -datastore** <directory> **-language** vb **-log** <log-name> [options] **-playback** <script-name> [**-args** <values>]
Play back a script, with an associated dataset	**rational_ft.exe -datastore** <directory> **-language** vb **-iterationCount** <count> **-playback** <script-name> **Note:** To playback your scripts with dynamic VPs, add `-rt.interative true` before `-playback` in the command line.
Construct an empty script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-map** <sharedmapname> [options] **-create** <script-name>
Regenerate the helper file for a script	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-helper** <script-name>
Regenerate all helper files for a datastore	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-regenHelpers** <script-name>
Display a object-map file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-display** <object-map filename>

Task	Command syntax
Display a verification-point file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-display** <verification point filename>
Edit a object-map file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-edit** <object-map filename>
Edit a verification-point file	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-language** vb **-edit** <verification point filename>
Create and edit a shared-object map	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-fromMap** <object-map filename1> **-createMap** <object-map filename2>
Merge a later version of an object map into a current (modified) version of the same map	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-from** <object-map filename1> **-to** <object-map filename2> **-original** <object-map filename1> **-mergeMap**
Compare an actual verification point result to an expected verification point result	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-datastore** <directory> **-baseline** <baseline verification point filename> **-compare** <expected verification point filename> <actual verification point filename>
Enable a configured browser, Java™ environment, or Eclipse platform	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-language** vb **-enableName** <browser/Java environment/Eclipse>
Disable all configured browsers, Java™ environments and Eclipse platforms	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-disableall**
Run the application configuration tool	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-language** vb **-appConfig** <application name>
Run the Test Object Inspector	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-language** vb **-inspector**
Run the Object Properties Configuration Tool	java **-classpath** <classpath> com.rational.test.ft.rational_ft **-language** vb **-objectlibrary**

Test application domain support

Rational® Functional TesterHCL OneTest™ UI is an object-oriented automated testing tool that tests Java™, HTML, .NET, Windows®, SAP, and Siebel applications.

Rational® Functional TesterHCL OneTest™ UI recognizes two different kinds of nesting of objects, a parent-child nesting and an owner-owned nesting. A parent-child nesting occurs when one object is contained within another, such as a button on a form. An owner-owned nesting occurs when the owned object has its own top-level window, such as a dialog box that is owned by a top-level window.

Rational® Functional TesterHCL OneTest™ UI supports some instances of nesting of objects from different domains. That means you can test an object of one domain that is nested inside an object of another domain, and Rational® Functional TesterHCL OneTest™ UI will accurately understand the objects and their domains. If a nesting of one domain within another is not supported, Rational® Functional TesterHCL OneTest™ UI will model the objects consistently but may not accurately understand the domain for the nested objects. For example, Rational® Functional TesterHCL OneTest™ UI does not support the nesting of a .Net control within a Windows® application. In this case, Rational® Functional TesterHCL OneTest™ UI is likely to see the .Net controls as if they were Windows® controls (because .Net controls are often implemented using the underlying mechanisms of Window controls). Another example is a Windows-based dialog that appears on top of a Java™ application. In this case, the Java™ domain does not understand the windows dialogs, or even acknowledge that they exist. And since by default Rational® Functional TesterHCL OneTest™ UI does not dynamically enable a Java™ application as if it were a Windows® application, there is no Windows® domain in that process. So in this instance, the dialogs are not testable objects without scripting.

The following list describes the cases of nested domains that are supported.

HTML - ActiveX as child -- This is an HTML page that contains ActiveX controls.

HTML - Windows® Owned -- Some common dialogs displayed by Internet Explorer will appear as Windows® domain objects. Note that these dialogs have no reasonable Mozilla equivalent and will therefore produce a script that is not cross-browser.

VB.Net - ActiveX as child -- This is a VB.Net Winforms application that utilizes legacy ActiveX controls.

VB.Net - HTML as child -- This is a VB.Net WinForms application that utilizes an embedded Internet Explorer browser control.

VB.Net - HTML as owned -- This is VB.Net WinForms application that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

VB.Net - Windows® as owned -- This a VB.Net WinForms application that displays some form of non-WinForm dialog. For example, when a VB.Net application displays a common dialog (File Open, Print, etc.) or messagebox.

Java™ - HTML as child -- This a Java™ SWT application that utilizes an embedded Internet Explorer browser control.

Java™ - HTML as owned -- This a Java™ SWT application that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

Windows® - HTML as child -- This is a generic Windows® application (possibly VB 6.0 or MFC) that utilizes an embedded Internet Explorer browser control.

Windows® - HTML as owned -- This is a generic windows application (possibly VB 6.0 or MFC) that utilizes an embedded Internet Explorer browser control that in turn displays a dialog composed of HTML (shown by calling ShowModalDialog in JavaScript™).

Nested domains in the object map

You can see that objects are of different test domains by looking at the object map. In the object map, each object is listed by its object type and domain type. For example, the following object:

Java™: Button: close-order: javax.swing.JButton

is a Java™ button that is a javax.swing.JButton object type. The "Java™:" prefix shows that the object is in the Java™ test domain. The test domain of every object is always the first thing shown on each object listed in the map, as shown in the example above. If a child object has a different domain than the parent object, you will see two different prefixes in their entries in the object map.

Add New Item dialog box

You use the Add New Item dialog box to add a script, an empty script, a test object map, or a test dataset to the currently selected project.

The Add New Item dialog box has the following controls:

Categories

Lists the project item category, Functional Test project Items.

Templates

Lists the project item types, which include Record Functional Test Script on page 902, Test Object Map, Functional Test Script on page 880, Script Helper Base Class, and Test dataset on page 938.

Name

Enter the name of the file you want to create. For a script, the name must be a valid VB.NET class name.

Location

Displays the current path of the new file. Enter a path or click **Browse** to select a path.

To open:

- Click the **Record a Functional Test Script** button on the product toolbar or select **File > New > Add Script Using Recorder** from the Rational® Functional TesterHCL OneTest™ UI menu.
- Click the **Create an Empty Functional Test Script** button on the product toolbar or select **File > New > Add Empty Script** from the product menu.
- Click the **Create a Test Object Map** button on the product toolbar or select **File > New > Add Test Object Map** from the product menu.
- Click the **Create a Test dataset** button on the product toolbar or select **File > New > Add Test dataset** from the product menu.

dataset Literal Substitution dialog box

You can use the dataset Literal Substitution dialog box to find or replace literal values in a test script with a dataset reference (an associated dataset). You can set the options in this dialog box to find and replace all, number, string, or boolean literals in a script with a dataset reference.

You can also add a literal from a script to a dataset. If you do not use an existing dataset variable, the same literal values (the values that were captured when you recorded the test script) are used each time that you run the script.

Literal Type

Click to set the type of literal that you want to find; either all, number, string, or boolean literals in a script. A literal is a letter or symbol that stands for itself as opposed to a feature, function, or entity associated with it in a programming language: $ can be a symbol that refers to the end of a line, but as a literal, it is a dollar sign. This setting works with the **Search up** setting. Click **Find Next** or **Replace** to move to the next type of literal in a script starting from the insertion point of your cursor.

All Types

Click to find all literals in a script.

Numbers

Click to find number literals in a script. A number includes integers (a whole number, not a fractional number, that can be positive, negative, or zero)or floating numbers (positive and negative decimal numbers).

Strings

Click to find string literals in a script. A string stores alphanumeric values such as name, city, or state.

Booleans

Click to find boolean literals in a script. Any use of the boolean literals true or false are flagged for substitution.

Search up

Click to move up through script to the next literal in the script from the insertion point. Clear to move forward through the script to the next literal. This setting works with the **Literal Type** setting, and with the **Find Next** or **Replace** settings.

Literal

Displays the name of a selected literal in a script.

dataset Variable

Type the name of the dataset variable with which you want to replace a literal or click the **dataset Variable** arrow to select the existing dataset variable that you want the script to reference. The default type for a new dataset column is string. You can change the variable type on page 1739 later.

Add Literal to dataset

Click to add a selected literal to a dataset variable when you click **Replace**. The selected literal is added to the dataset in the variable that you specify in **dataset Variable**. A dataset reference replaces the selected literal when you click **Replace**.

Find Next

Click to find the next literal in a script. This setting works with **Search up** and **Literal Type** settings.

Replace

Click to replace the selected literal with a dataset reference. The dataset reference is made to the dataset variable that you specify in **dataset Variable**. When you click **Add literal to dataset**, Functional Test adds a selected literal to the dataset variable that you specify in **dataset Variable** . Automatically moves to the next literal in the test script after replacing a literal. This setting works with **Search up** and **Literal Type** settings.

Close

Click to close the dialog box without finding a literal or replacing a literal with a dataset reference.

To open: Click **Script> Find Literals and Replace with dataset Reference**.

Related Topics:

Replacing Literals with dataset References on page 955

Associating a dataset with a Test Script on page 953

Development Environment options

You use the Development Environment page to control the behavior of the Development Environment while playing back, recording, and debugging Functional Test scripts.

The Development Environment page has the following controls:

Development Environment state during run

Controls the Development Environment display while playing back scripts.

Minimized

Reduces the Development Environment to a button on the task bar during playback.

Minimized and restored on playback termination (Default)

Reduces the Development Environment to a button on the task bar during playback and restores it when playback finishes.

Hidden

Hides the Development Environment during playback and restores it when playback finishes.

Leave in current state

Does not change the Development Environment during playback.

Development Environment state during recording

Controls the Development Environment display while recording scripts.

Minimized

Reduces the Development Environment to a button on the task bar during recording.

Minimized and restored when recording finished (Default)

Reduces the Development Environment to a button on the task bar during recording and restores it after recording stops.

Hidden

Hides the Development Environment during recording and restores it after recording stops.

Leave in current state

Does not change the Development Environment during recording.

Development Environment state during debug

Controls the Development Environment display while debugging scripts.

Minimized

Reduces the Development Environment to a button on the task bar during debugging.

Minimized and restored on playback termination

Reduces the Development Environment to a button on the task bar during debugging and restores it after debugging stops.

Hidden

Hides the Development Environment during debugging and restores it after debugging stops.

Leave in current state (Default)

Does not change the Development Environment during debugging.

Restore Defaults

Restores the default values on this page.

Apply

Saves your changes without closing the dialog box.

To open: Click **Tools > Options**. In the left pane, expand **Functional Test** and click **Development Environment** or **General**.

Related Topics:

Advanced Development Environment Options

Setting Functional Test Options

About Functional Test Options

Menus

This topic describes the product menu as it appears in Visual Basic when you have Rational® Functional TesterHCL OneTest™ UI installed. Menus and menu items on your machine might be different, depending on the version of Visual Studio .NET that is installed, the installation options selected, and what is open in Visual Studio.

The product menu has the following options:

- **File** on page 1902
- **Edit** on page 1903
- **View** on page 1904
- **Project** on page 1904
- **Build** on page 1906
- **Debug** on page 1906
- **Script** on page 1906
- **Configure** on page 1907
- **Tools** on page 1907
- **Window** on page 1907
- **Help** on page 1907

File menu

The **File** menu has the following commands:

New -- Opens the appropriate dialog box to create a new project, new file, blank solution, and a new Functional Test project. You can also record a Functional Test script and a Rational® Functional TesterHCL OneTest™ UI empty script, and create a test object map and a test dataset. For more information, see the Visual Studio .NET online help.

Open -- Opens the appropriate dialog box to open a project, a project from the web, a file, and a file from the web. You can also convert a Java™ Language project to an equivalent Visual C# .NET project or convert a Visual Basic 6.0 project to an equivalent Visual Basic .NET project. For more information, see the Visual Studio .NET online help.

Close -- Closes the script currently displayed in the Code Editor.

Add New Item -- Opens the Add New Item dialog box that enables you to add an item to the currently selected project. For more information, see the Visual Studio .NET online help.

Add Existing Item -- Opens the Add Existing Item dialog box, which enables you to select individual or multiple files to add to your project. For more information, see the Visual Studio .NET online help.

Add Project -- Opens the appropriate dialog box that enables you to add a new project, an existing project or an existing project from the web. For more information, see the Visual Studio .NET online help.

Open Solution -- Opens the Open Solution dialog box, which enables you to open an existing solution (.sln) file. For more information, see the Visual Studio .NET online help.

Close Solution -- Closes the current solution.

Save *script* -- Saves your changes you made to the script currently open in the Code Editor.

Note: *name* As -- Opens the Save File As dialog box, which enables you to save the current file under another name and location. **Note:** Do not use this option to save scripts.

Advanced Save Options -- Opens the Advanced Save Options dialog box, which enables you to specify non-default language encoding options and to register your line-ending character preferences for an open file. For more information, see the Visual Studio .NET online help.

Save All -- Saves all scripts and files currently open.

Page Setup -- Opens the Page Setup dialog box.

Print -- Opens the Print dialog box.

Recent Files -- Opens the names of project items that you have used most recently . For more information, see the Visual Studio .NET online help.

Recent Projects -- Opens the names of projects you have used most recently. For more information, see the Visual Studio .NET online help.

Exit -- Closes Rational® Functional TesterHCL OneTest™ UI.

Edit menu

The **Edit** menu has the following options:

Undo -- Cancels the last edit you made, if possible. Some edits cannot be undone.

Redo -- Does the action of the last Undo action again.

Cut -- Deletes the selected characters and puts them on the Clipboard.

Copy -- Copies the selected characters to the Clipboard.

Paste -- Inserts at the cursor any characters that were previously cut or copied to the Clipboard.

Cycle Clipboard Ring -- Cycles through the contents of the Clipboard Ring tab. For more information, see the Visual Studio .NET online help.

Delete -- Removes the selected characters on the Code Editor or the selected item in the Solution Explorer.

Select All -- Selects all the characters in the currently open script or file.

Find and Replace -- Opens the appropriate dialog box to enable you to search for and substitute text in the current document or window and within multiple documents. For more information, see the Visual Studio .NET online help.

Go to -- Opens the Go to Line dialog box, which enables you to move to a line that you specify in the currently active script or file.

Insert File as Text -- Inserts the contents of the selected file, such as text or code, for use in your application. For more information, see the Visual Studio .NET online help.

Advanced -- Provides several formatting options. For more information, see the Visual Studio .NET online help.

Bookmarks -- Provides several options for working with bookmarks. For more information, see the Visual Studio .NET online help.

Outlining -- Provides several options for creating a hierarchical, outlined overview that gives you a view of the structure of your document. For more information, see the Visual Studio .NET online help.

IntelliSense -- Provides an array of options that make language references easily accessible. When coding, you do not need to close the text editor to perform searches on language elements. You can preserve your context, find the information you need, and insert language elements directly into your code. For more information, see the Visual Studio .NET online help.

View menu

The **View** menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Script Explorer, Test dataset, and Solution Explorer menu items. For more information, see the Visual Studio .NET online help.

Script Explorer -- Starts the Script Explorer with a list of the script helper, helper base class, test dataset, verification points, and test objects for the current script.

Test dataset -- Opens the test dataset editor.

Solution Explorer -- Starts the Solution Explorer, which provides a structured view of your projects and their files and access to the commands that pertain to them. For more information, see the Visual Studio .NET online help.

Project menu

The **Project** menu has the following options:

Add Script Using Recorder -- Opens the Add New Item dialog box, which enables you to enter information about the new script and start recording on page 902.

Add Empty Script -- Opens the Add New Item dialog box, which enables you to create a script on page 880 you can use to manually add Java™ code.

Add Test Object Map -- Opens the Add New Item dialog box, which enables you to add a new test object map to a project.

Add Test dataset -- Opens the Add New Item dialog box, which enables you to add a new test dataset on page 943 to a project.

Add Helper Base Class -- Opens the Add New Item dialog box, which enables you to add a new script helper base class.

Add Windows Form -- Opens the Add New Item dialog box, which enables you to add a form for Windows® applications to the currently selected project. For more information, see the Visual Studio .NET online help.

Add Inherited Form -- Opens the Add New Item dialog box, which enables you to add a new form based on an existing Windows® form. For more information, see the Visual Studio .NET online help.

Add User Control -- Opens the Add New Item dialog box, which enables you to add a new class for creating a Windows® Form control using the visual designer. For more information, see the Visual Studio .NET online help.

Add Inherited Control -- Opens the Add New Item dialog box, which enables you to add a new control based on an existing Windows® Form control. For more information, see the Visual Studio .NET online help.

Add Component -- Opens the Add New Item dialog box, which enables you to add a class for creating components using the visual designer. For more information, see the Visual Studio .NET online help.

Add Module -- Opens the Add New Item dialog box, which enables you to add a file for storing groups of functions. For more information, see the Visual Studio .NET online help.

Add Class -- Opens the Add New Item dialog box, which enables you to add an empty class declaration. For more information, see the Visual Studio .NET online help.

Add New Item -- Opens the Add New Item dialog box, which enables you to add a form for Windows® applications. For more information, see the Visual Studio .NET online help.

Add Existing Item -- Opens the Add Existing Item dialog box, which enables you to select individual or multiple files to add to your project. For more information, see the Visual Studio .NET online help.

Exclude From Project -- Temporarily excludes the selected item from the project. For more information, see the Visual Studio .NET online help.

Show All Files -- Restores a file to the project build that you previously excluded.

Add Reference -- Opens the Add Reference dialog box, which enables you to list the various types of components and projects you can browse. For more information, see the Visual Studio .NET online help.

Add Web Reference -- Opens the Add Web Reference dialog box, which enables you to browse the web for existing web services and to add web references to your application. For more information, see the Visual Studio .NET online help.

Build menu

The **Build** menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI. For more information, see the Visual Studio .NET online help.

Debug menu

The **Debug** menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI. For more information, see the Visual Studio .NET online help.

Script menu

The **Script** menu has the following options:

Run -- Compiles and runs the script on page 1445 currently displayed in the Code Editor.

Debug -- Compiles and debugs the script currently displayed in the Code Editor.

Add Script Using Recorder -- Opens the Add New Item dialog box, which enables you to enter information about the new script and start recording on page 902.

Insert Recording -- Starts recording at the cursor location on page 881 in the current script, which enables you to start applications, insert verification points, and add script support functions.

Open Test Object Map -- Opens the test object map on page 1838 associated with the current script.

Update Script Helper -- Updates the helper file (*ScriptHelper.vb) to reflect changes made to the template for selected test asset.

Insert Test Object -- Opens the Insert a GUI Object into the Object Map wizard, which enables you to add objects to the test object map and the script.

Insert Verification Point -- Starts the Verification Point and Action Wizard on page 1844, which enables you to insert a verification point in the current script at the cursor location.

Insert Data Driven Commands -- Opens the Data Drive Actions page of the dataset Population Wizard to select the objects in an application-under-test to data-drive an application on page 938.

Find Literals and Replace with dataset Reference -- Replaces literal values with a dataset reference in a test script to add realistic data to an existing test script on page 955.

Configure menu

The **Configure** menu has the following options:

Configure Applications for Testing -- Starts the Application Configuration Tool on page 1735, which enables you to add and edit configuration information on page 796 -- such as name, path, and other information that Rational® Functional TesterHCL OneTest™ UI uses to start and run the application -- for the Java™ and HTML applications you want to test.

Enable Environments for Testing -- Opens the Enable Java™ Environments (JRE)/Web Browsers/Eclipse Platforms for Testing dialog box, which you use to enable Java environments on page 778, browsers on page 780, and Eclipse platforms on page 1756 and to configure them.

Tools menu

The **Tools** menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Test Object Inspector, Synchronize Solution with Editor, and Options menu items. See Visual Studio .NET online help for more information.

Test Object Inspector -- Opens the Test Object Inspector window to dynamically view properties of selected test objects visible in the running application and displays information about those objects, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information.

Synchronize Solution with Editor -- Synchronizes the solution with the Code Editor.

Options -- Opens the Options dialog box, which enables you to customize Rational® Functional TesterHCL OneTest™ UI by setting preferences for Rational® Functional TesterHCL OneTest™ UI, Highlight, Operating System, Playback, Recorder, Development Environment, and Advanced Development Environment.

Window menu

The **Window** menu contains Visual Studio menu items that are not applicable to Rational® Functional TesterHCL OneTest™ UI except for the Recent Scripts and Window menu items.

Recent Scripts -- Displays the names of scripts you have most recently used.

Windows -- Opens the Windows® dialog box, which enables you to display, close, and arrange documents that are open in editors and designers. For more information, see the Microsoft® Document Explorer Help.

Help menu

The **Help** menu has the following product menu items:

Dynamic Help -- Displays links to topics related to your current work in the Dynamic Help window. The links reflect the portion of the integrated development environment that currently has focus.

Contents -- Opens the table of contents in the Contents window.

Index -- Opens a list of keywords, arranged alphabetically, in the Index window.

Search -- Opens the Search window, which enables you to do locate all Library topics for a particular word or phrase

Index results -- Opens the Index Results window, which lists the topics associated with the keyword you selected in the Index window. The Index Results window opens if the keyword you selected has more than one topic associated with it.

Search results -- Opens the Search Results window, which displays the topics that include matches to the text you entered in the Search window.

Edit Filters -- Opens the Edit Help Filters window, which enables you to change edit filter definitions. For more information, see the MSDN Library Viewer online help.

Previous topic -- Navigates to the previous topic in the table of contents.

Next topic -- Navigates to the next topic in the table of contents.

Sync Contents -- Synchronizes the table of contents with the document window.

Show Start Page -- Displays the Start Page, which provides easy access to recent projects and enables you to quickly create a new project.

Check for Updates -- Checks for available Service Releases.

Technical Support -- Provides links to get support information based on the software and associated Help collections that were provided to your computer during software installation.

Help on Help -- Provides information about using the online Help collections.

About Microsoft Visual Basic .NET -- Displays information about the current version of Microsoft® Visual Basic .NET you are running.

Functional Test project properties page

Use this page to change your default script helper base class.

Rational® Functional TesterHCL OneTest™ UI uses the helper base class for all the scripts you create in this project.

You can change the base class for a script by specifying it in the Functional Test Script Properties Page on page 1909.

By default, all Functional Test scripts extend the RationalTestScript class and inherit a number of methods (such as callScript). You can create your own helper base class, which can add methods or override the methods from RationalTestScript. Use this properties page to change the default helper base class for a project.

Default Script Helper Base Class -- Enter the fully qualified class name of your custom helper base class in this field. Note that your helper base class must extend RationalTestScript.

If you change your base class and want to reset it to RationalTestScript, you can either type RationalTestScript in the **Default Script Helper Base Class** field or clear the field. Leaving this field blank resets the script to use RationalTestScript.

To open: In the Projects view, select a project, right-click, and click **Properties**. Click **Functional Test Project**.

Related Topics:

About Changing the Default Script Helper Base Class

Changing the Default Script Helper Base Class for a Project

Changing the Default Script Helper Base Class for a Script

Functional Test script properties page

Use this page to change your script helper base class, test dataset, or dataset iteration for an individual script.

Rational® Functional TesterHCL OneTest™ UI uses a default helper base class for all the scripts you create in a project. This base class is listed in the Functional Test Project Properties Page on page 1908. You can use a different base class for an individual script by changing it in this Script Properties Page.

By default, all Functional Test scripts extend the RationalTestScript class and inherit a number of methods (such as callScript). You can create your own helper base class, which can add additional methods or override the methods from RationalTestScript. Use this properties page to change the default helper base class for the selected script.

Test Object Map -- Displays either **Private** to indicate that the script test object map is private or the name of the shared test object map.

Helper Base Class -- Enter the fully qualified class name of your custom helper base class in this field. Note that your helper base class must extend RationalTestScript.

If you change your base class and want to reset it to RationalTestScript, you can either type RationalTestScript in the **Helper Base Class** field or clear the field. Leaving this field blank resets the script so that it uses RationalTestScript.

Test dataset -- Click **Browse** to change the dataset associated with a script.

dataset Record Selection Order -- Specifies how a test script accesses records in its associated dataset when you play back the test script. Click the **dataset Record Selection Order** arrow to change the test dataset iterator.

Types of dataset iterators:

- **Sequential** -- Makes a test script access records in the dataset in the order that they appear in the dataset. This is the default dataset iterator.

- **Random** -- Makes a test script access records in the dataset randomly. A random iterator accesses every record in the dataset once.

To open: In the Solution Explorer, select a script, right-click, and click **Properties**.

Related Topics:

About Changing the Default Script Helper Base Class

Changing the dataset Iterator on page 953

Changing the Default Script Helper Base Class for a Project

Changing the Default Script Helper Base Class for a Script

Select Helper Superclass/Select a Script Helper Base Class Dialog Box on page 1829

Toolbar

This topic describes all the buttons and controls on the Rational® Functional TesterHCL OneTest™ UI toolbar.

The product toolbar has the following buttons and controls:

Note: This topic describes the Rational® Functional TesterHCL OneTest™ UI menu as it appears in Visual Basic when you have Rational® Functional TesterHCL OneTest™ UI installed. Menus and menu options on your machine might be different depending on the version of Visual Studio .NET you have installed, the installation options you have selected, and what is open in Visual Studio.

New Project -- Displays the appropriate dialog box that enables you to create a new project or solution. Click to select **New Project** or **New Blank Solution**. See Visual Studio .NET online help for more information.

Add New Item -- Displays the Add New Item dialog box, which enables you to add an item to the currently selected project. Click to display the list of items to create. See Visual Studio .NET online help for more information.

Open File -- Displays the Open File dialog box, which enables you to open an existing file from disk. You can also use this dialog box to open an already opened file using different language encoding options. See Visual Studio .NET online help for more information.

Save *script* -- Saves any changes you made to the script currently displayed in the Code Editor.

Save All -- Saves all the scripts and files currently open.

 **Cut** -- Deletes the highlighted characters and puts them on the Clipboard.

 **Copy** -- Copies the highlighted characters to the Clipboard.

 **Paste** -- Inserts at the cursor any characters that were previously cut or copied to the Clipboard.

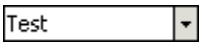
 **Undo** -- Cancels the last edits you made, if possible. Some edits cannot be undone. Click ▾ to display the list of actions to undo.

 **Redo** -- Re-executes the last action that was undone by the Undo button or the user. Click ▾ to display the list of actions to redo.

 **Navigate Backward** -- Enables you to navigate through the open windows in the environment as well as the selection and cursor history within files. Click ▾ to list open windows. See Visual Studio .NET online help for more information.

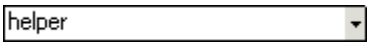
 **Navigate Forward** -- Enables you to navigate through the open windows in the environment as well as the selection and cursor history within files. Click ▾ to list open windows. See Visual Studio .NET online help for more information.

 Start -- Runs the build process, loads the debugger, and opens the browser to view the content. Launches the current script and displays the Debug perspective, which provides information as the script debugs. For information, see Debugging Scripts on page 1449.



Solution Configurations -- Displays the available solution build configurations. You can also access the Configuration Manager from this drop-down list. See Visual Studio .NET online help for more information.

 **Find in Files** -- Displays the Find in Files dialog box, which enables you to search within various files and review search results in report format. See Visual Studio .NET online help for more information.



Find -- Displays up to the last 16 search items entered.

 **Solution Explorer** -- Displays the Solution Explorer, which provides with an organized view of your projects and their files as well as access to the commands that pertain to them. See Visual Studio .NET online help for more information. See Visual Studio .NET online help for more information.

 **Properties Window** -- Displays the Properties window, which you can use to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the Properties window to edit and view file, project, and solution properties. See Visual Studio .NET online help for more information.

 **Object Browser** -- Displays the Object Browser, which enables you to browse, filter, and search for components that you might want to use in your projects. See Visual Studio .NET online help for more information.

 **Toolbox** -- Displays a variety of items (such as .NET components, COM components, HTML objects, code fragments, and text) for use in Visual Studio projects. The tabs and items available from the Toolbox change, depending on the designer or editor currently in use. See Visual Studio .NET online help for more information.

 **Class View** -- Displays the Class View, which provides a programmatic view of symbols in your code, for example, namespaces, classes, methods, and functions. Click to list other IDE components to display. Use the See Visual Studio .NET online help for more information.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

 **Create a Functional Test project** -- Displays the New Project dialog box, which enables you to create a new Functional Test project.

 **New Folder** -- Creates a new folder in the Solution Explorer.

 **Create an Empty Functional Test Script** -- Displays the Add New Item dialog box, which enables you to create an empty script on page 880 you can use to add code.

 **Create a Test dataset** -- Displays the Add New Item dialog box, which enables you to Creating a dataset on page 943.

 **Create a Test Object Map** -- Displays the Add New Item dialog box, which enables you to add a new test object map to a project.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

 **Record a Functional Test Script** -- Displays the Add New Item dialog box, which enables you to enter information about the new script and start recording on page 902.

 **Insert Recording into Active Functional Test Script --** Starts recording at the cursor location in the current script, which enables you to start applications, insert verification points, and add script support functions.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

 **Run Functional Test Script** -- Plays back the Functional Test script currently displayed in the Code Editor. For information see Running Scripts from Rational® Functional TesterHCL OneTest™ UI on page 1445.

Debug Functional Test Script -- Launches the current script and displays the Debug perspective, which provides information as the script debugs. For information, see Debugging Scripts on page 1449.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

Configure Applications for Testing -- Displays the Application Configuration Tool on page 1735, which enables you to add and edit configuration information -- such as name, path, and other information that Functional Test uses to start and run the application -- for the Java™ and HTML applications you want to test.

Enable Environments for Testing -- Displays the Enable Java™ Environments (JRE)/Web Browsers/Eclipse Platforms for Testing dialog box, which you use to enable Java environments, on page 778 configure applications, on page 796 and browsers on page 780 to configure your JRE on page 797s and browsers on page 798.

Open Test Object Inspector -- Displays the Test Object Inspector tool, which enables you to display test object information, such as parent hierarchy, inheritance hierarchy, test object properties, nonvalue properties, and method information.

Synchronize Solution with Editor -- Scrolls in the tree hierarchy to the name of the script currently displayed in the Code Editor.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

Insert Test Object into Active Functional Test Script -- Displays the Select an Object dialog box, which enables you to select test objects to add to the test object map and a script.

Insert Verification Point into Active Functional Test Script -- Displays the Select an Object page of the Verification Point and Action wizard, which enables you to select an object in your application you want to perform a test on. For information, see Inserting a Verification Point from the Script Explorer.

Insert Data Driven Commands into Active Functional Test Script -- Displays the Data Drive Actions page of the dataset Population Wizard, which enables you to select the objects in an application-under-test to data-drive an application.

Find Literals and Replace with dataset Reference -- Replaces literal values with a dataset reference in a test script, which enables you to add realistic data to an existing test script.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

Display an object Member List based on the current Language -- Displays a list of valid member variables or functions for the appropriate class, struct, union or namespace, Selecting from the list inserts the member into your code. See Visual Studio .NET online help for more information.

 **Display Parameter Info based on the current language** -- Displays a list of information about the number, names, and types of parameters required by a function, template, or attribute. See Visual Studio .NET online help for more information.

 **Display Quick Info based on the current language** -- Displays the complete declaration for any identifier in your code when the mouse cursor hovers over an identifier. See Visual Studio .NET online help for more information.

 **Display Word Completion based on the current language** -- Once you enter enough characters to uniquely identify an object or keyword, provides the rest of the word. See Visual Studio .NET online help for more information.

 **Decrease Indent** -- Removes one level of indent for one or more paragraphs.

 **Increase Indent** -- Indents the selected paragraph by one indent unit.

 **Comment out the selected lines** -- adds a single quote marker or the keyword REM to the selected line to indicate that the line is a comment and should not be processed.

 **Uncomment the selected lines** -- Removes the single quote marker or the keyword REM from the selected line.

 **Toggle an unnamed bookmark on the current line** -- Adds a temporary bookmark to the selected line. See Visual Studio .NET online help for more information.

 **Move the caret to the next unnamed bookmark** -- Returns to the next temporary bookmark. See Visual Studio .NET online help for more information.

 **Move the caret to the previous unnamed bookmark** -- Returns to the previous temporary bookmark. See Visual Studio .NET online help for more information.

 **Clear all unnamed bookmarks in the current document** -- Removes temporary bookmarks in a document. See Visual Studio .NET online help for more information.

Toolbar Options -- Enables you to display/hide toolbars on the IDE. See Visual Studio .NET online help for more information.

Options dialog box

You use the Options dialog box to customize Rational® Functional TesterHCL OneTest™ UI in a number of different areas, such as increasing/decreasing all time options and setting options for highlighting test objects; operating system values; playback; delays; logging; playback monitor; ScriptAssure(TM); recorder; Recording Monitor; and

color settings for the Verification Point Editor, Verification Point Comparator, and the Object Map Editor; and the development environment

The Options dialog box has the following tabs. Click the tab name below for information about how to use that tab.

General Rational® Functional TesterHCL OneTest™ UI on page 828

Increases or decreases all Rational® Functional TesterHCL OneTest™ UI time options.

Highlight on page 831

Specifies how to emphasize test objects when you select them in applications-under-test. You can specify color, width, speed, and time.

Operating System on page 836

Sets values that are operating system dependent.

Playback on page 836

Sets Functional Test script playback options.

Delays on page 839

Sets delays during Functional Test script playback. These settings are useful to slow down the rate at which script commands are sent to the operating system.

Logging on page 834

Use to indicate log viewing and Comparator options.

Monitor on page 840

Use to specify whether to display the playback monitor during playback.

ScriptAssure(TM) on page 841

Use the Standard Options page to control object-matching sensitivity during play back. You can use the Advanced Options page on page 840 to set thresholds for recognition scores, which Functional Test uses when searching for objects during script playback.

Recorder on page 843

Use to indicate options for recording Functional Test scripts.

Monitor on page 844

Use to change colors without going to the Recording Monitor. Monitor page buttons reflect the current settings in the Recording Monitor.

UI Color on page 830

Use to specify color settings for the Verification Point Editor, Verification Point Comparator, and the Object Map Editor.

Development Environment on page 1900

Use to control the behavior of the Development Environment while playing back, recording, and debugging Functional Test scripts.

Advanced

Use to control warning messages when the TestManager log cannot be used.

To open: Click **Tools > Options** and in the left pane, expand **Functional Test**.

Related Topics:

Setting Rational® Functional TesterHCL OneTest™ UI Options

About Rational® Functional TesterHCL OneTest™ UI Options

Java API reference

You can find the links to view the following API reference guides of IBM® Rational® Functional TesterHCL OneTest™ UI:

- Java API helpJava API help
- Java Proxy API helpJava Proxy API help

Security Considerations

You can take certain actions to ensure that your installation is secure, customize your security settings, and set up user access controls.

Security Considerations for IBM® Rational® Functional TesterHCL OneTest™ UI

This document describes the actions that you can take to ensure that your installation is secure, customize your security settings, and set up user access controls.

Ports, protocols, and services

Rational® Functional TesterHCL OneTest™ UI uses port number 9100, by default, to launch a local web server to provide browser support. This port accepts non-secure communication and is an open port. The port is configurable. The enablement process for Java and browsers must run with administrator or super user credentials.

Customizing your security settings

datasets can be encrypted and access controlled by passwords that are difficult, but not impossible, to break. Logging in to IBM® Rational® Quality Manager server either through the adapter or the Keyword View requires a username and password and this information is encrypted by using 128-bit encryption and is stored locally in encrypted form.

Privacy policy considerations

This software offering does not use cookies or other technologies to collect personally identifiable information.

Security limitations

Passwords that are entered as a part of a test script recording are stored in the test script as plain text.

Notices

This document provides information about copyright, trademarks, terms and conditions for the product documentation.

© Copyright IBM Corporation 2001, 2016 / © Copyright HCL Technologies Limited 2016, 2023

This information was developed for products and services offered in the US.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM®HCL® Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

INTERNATIONAL BUSINESS MACHINES CORPORATIONHCL TECHNOLOGIES LTD. 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 jurisdictions 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®HCL® 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®HCL® websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM®HCL® product and use of those websites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM®HCL® under terms of the IBM®HCL® Customer Agreement, IBM®HCL® International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM®HCL® products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM®HCL® has not tested those products and cannot confirm

the accuracy of performance, compatibility or any other claims related to non-IBM®HCL® products. Questions on the capabilities of non-IBM®HCL® products should be addressed to the suppliers of those products.

Statements regarding IBM®HCL®'s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 actual people or business enterprises 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®HCL®, 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®HCL®, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM®HCL® shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:
© (your company name) (year).
Portions of this code are derived from IBM Corp. HCL Ltd. Sample Programs.
© Copyright IBM Corp. 2001, 2016HCL Ltd. 2016, 2020.

Trademarks

IBM®HCL®, the IBM®HCL® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM® or other companies. A current list of IBM® trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

IBM®HCL®, the IBM®HCL® logo, and hcl.com® are trademarks or registered trademarks of HCL Technologies Ltd., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM®HCL® or other companies.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM®HCL® website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM®HCL®.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM®HCL®.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM®HCL® reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM®HCL®, the preceding instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM®HCL® MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (Software Offerings) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offerings use of cookies are the following:

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBMs Privacy Policy at http://www.ibm.com/privacy and IBMs Online Privacy Statement at http://www.ibm.com/privacy/details in

the section entitled Cookies, Web Beacons and Other Technologies, and the IBM Software Products and Software-as-a-Service Privacy Statement at http://www.ibm.com/software/info/product-privacy.

Index

Special Characters

.Net 1721

A

accessibility 845
  mapping keyboard shortcut keys 844
actions
  data driven tests 1781
actions on objects
  verification points during
  recording 1820, 1847
  verification points without
  recording 1820, 1847
active windows
  unexpected
    handling 1188
ActiveX 1721
adapters
  Rational Quality Manager 358, 362, 365
add control 874
add object 1884
Add user action for this element 596
adding 802
administrator 201
Adobe Flex application 175
adobe pdf 1704
AJAX
  support
    AJAX support 804
    AJAX Support 1708
    autotrace 804
    existing scripts 805
ambiguous control 871
API 1215
Appium
  importing Appium Java projects 734
  managing Appium tests 733
applets
  testing in HTML pages 1547
application
  objects
    adding to test object maps 1823
application view
  snapshots 1888
application visual 873, 873, 874, 934, 935
  verification points 934
application visuals
  snapshots 1888
applications
  configurations
    adding 1733
    testing 796, 1735
  object hightlighting 1836
  object selections 1821
  starting 896, 1837
ARM
  Application Response Measurement
    response 1708
    time 1708
assets
  testing shared assets 366
assign trust 811
AUT
  interaction 1086

B

base class
  functional test project properties 1908
  overview 888
  XDE Tester properties 1770
  XDE Tester script properties 1909
bookmarks
  view 1738
breakdown 1708
browsers
  configuring 798
  configuring for testing 1871
  enabling 780, 799, 1738, 1758
  functional test support 1715
  hard disk searches 1819
  Java plug-in enablement 800

C

callScript command 1739, 1816
CallScript method
  parameters 1200
change management 305
checkouts
  views in functional testing 1775
Chinese systems
  double byte characters 887
class view
  Visual Studio .NET 59
ClearCookies class 1056
clipboard
  assign text 895, 1741
  set text 895, 1741
  verification points 895, 1741
clock skew
  correcting 1486
code
  editor
    Visual Studio .NET 59
  samples 1190
colors
  changes 830
combination boxes
  data extraction in functional tests 1202
command line
  create config file 1400
  example 1730
  functional testing
    Visual Basic .NET scripting
    interface 1893
  schedule runs 1381
commands
  callScript 1739, 1816
  getProperty 885, 1782, 1850
  startApp 1837
  syntax 827
  waitForExistence 1784, 1851
comments
  functional testing
    scripts 895
Comparator
  color changes 830
  functional testing
    usage 923, 933, 1855
compile-time 814
compound tests
  adding tests 740
  adding to Test Workbench projects 743
  modifying 741
  overview 737
  running 741
  viewing 739
ComputerSpecific class 1058
configuration management
  software 305, 514
configurations
  applications 1733
  dynamic find 1440, 1441
  Java Runtime Environments 1819
  SAP 714
  test applications 796, 1735
  unexpected windows 842, 1438, 1743
  web browsers 1819
Configure applications
  creating 536
Configure Flex 135
configure object recognition properties 140
connection details
  SAP 725
consoles
  views
    functional testing 1744
control
  adding more data types 1103
  adding more properties 1101
  changing the mappability 1113
  changing the role 1110
  modify recognition property and
  weight 1111
controls
  object actions 1890
controls from different test domains
  test application domain support 1896
cookies
  setting and clearing for virtual
  users 1056
CountAllIterations class 1055
counter
  manage counters 1485
CountUserIterations class 1055
CSV files
  datasets
    data exportation 1759
  exporting to datasets 1780
CSV format
  exporting results 1489
Custom code
  debug 1069
custom counters
  test execution services 1064
custom Java code
  code execution counts 1055
  controlling loops 1048
  creating 1042
  custom counters 1064
  determining where a test is
  running 1058
  interfaces and classes 1044
  migrating 1074
  printing input arguments to a file 1054
  retrieving the maximum JVM heap
  size 1062
  retrieving virtual user IP address 1053
  running a program with a test 1062
  setting and clearing cookies for virtual
  users 1056
  statistics 1066
  transactions 1066
  using strings 1059
customization file 1115

customizing
Rational
Functional Tester
HCL
OneTest
UI
for accessibility
keyboard shortcuts 845
mapping keyboard shortcut keys 854
Customizing screen size 962

D

data
recognition
verification points 923, 1855
tests using realistic data 943
verification points
actions 1852
data correlation
description 661
disabling 675
manual 667
multiple fields 670
overview 661, 665
problem identification 672
regenerating tests 676
substituting 668
tests
data correlation 661
multiple-field data correlation 670
troubleshooting 675
viewing 664
data driven 944
data verification
programmatic screen scrapping 1006
data verification point 934
creating 907
data verification points
creating 108, 130, 1784, 1848
data-driving tests
data driving a test script 938
datasets 943
overview 937
test objects 1833
tutorial
playing back a test 174
tutorials 169
adding data to the dataset 172
creating a project and recording a script 168
creating a verification point with a dataset reference 171
data-driven test creation 166
descriptive headings 169
DataAreaLockException (test execution services) 1044
dataset 944
datasets 125
adding data 128, 131
associations removal 954
changing passwords 946
column changes 1739, 1758
column insertions 1735
creating in test 637
creating in workspace 640
data-driving scripts and 938
deleting 954
digital certificates 660
editing 653
encrypting 659, 946
encryption 658
exporting 1759
functional testing
empty 1746
importing from existing datasets 1780
literal value conversions 1751
literal value substitutions 1749, 1899
modifications 947
navigating to tests 660
new 128, 943
object options 1830
options 648
overview 636
record selection order 953
playback options 1882
select script assets 1834
references 130
literals 945, 950
removing encryption 659
script associations 953
segmented
row assignment 648
test references 648
test value associations 651
tutorials
data-driven test creation 166
types 943
typical 648
value references 955
verification point values 951
viewing in tests 652
DataWindow
PowerBuild 1217
Debug custom code 1069
default reports
changing 1485
defects
submitting 356
delays
settings 894
timers 1818
timers in tests 893
delete test line 872
Deploying extended log file 1503
deploying proxy 1119
describe function 1217
describe() function 1217
development enivornment
options 1900
development environment 134, 137, 175, 176
digital certificates
using with datasets 660
dll 802
dojo 1709
double byte characters
input method editors 887
dynamic help
Visual Studio .NET 59
dynamic test object
insert 1439

E

Eclipse applications
enabling 793, 1541
p2-based 793, 1541
Eclipse platforms
enabling for functional testing 1756
edit parts
testing GEF applications 139
edit simplified scripts
test lines 867
EditPart 1215
element labels
batch connection details 731
batch input transaction details 732
get (SAP) 728
SAP 729
screen details 726
server request details 730
set (SAP) 727
empty scripts
coding 879
Enable Flex application 134, 135, 175, 176
enabled Flex application 136, 137, 137, 177, 179
error handling
HTTP 676
error message color
recorder monitor preferences 844, 1809
errors 685, 1507
displaying views 1838
viewing 1494
examples
functional testing 198
exception 871, 1887
exception dialog 836, 837, 1796
exceptions 1892
ExecTest class 1062
execution variables
passing from Rational Quality Manager 495
Export
Event Console Output 1495
export objects 1886
export simplified scripts 877
exporting
item selections 1830
reports
to .view file 1490
to HTML format 1488
results
to CSV format 1489
extend
data driving 1109
extracting electronic images
electronic images 211

F

FAQ 684, 685, 1507, 1561
features flex custom control 1121
fields
data correlation 670
Figure 1215
files
printing input arguments to 1054
filters
searches of test objects 1752
find product packages 293
Flex
assign trust 811
command-line 815
compile application 815, 816, 816, 817
configure runtime loader 820
deploy 817
enable application 811, 815, 816, 816, 817
enable applications 813
enable compile-time 813
enable run-time 818
HTML wrapper 816
locally 818

overview 809, 1711
prerequisite 809
set up environment 811, 811
test 809
test application 811, 817, 818
test enabled applications 813
test non-enabled application 821
test non-enabled applications 818
web server 817
Flex application 176
Flex automation 135, 176
Flex Builder 816
Flex custom control support 1121, 1714
Flex developer 175
Flex support 809, 1711
Flex user interface 135
folders
creating new 1745
functional testing
new 862
fonts
changing 828
FTE 1726
functional test projects
creating 858
functional testing 135, 175, 176
command line
Visual Basic .NET scripting
interface 1893
datasets
empty 1746
examples 198
general options 1915
importing from existing projects 1830
integrations 501
keyword
view 500
menus 1762, 1902
Visual Studio .NET 59
perspectives 46
preference settings 822, 823
preferences 828, 1769, 1799
recording monitor 1792
previous version updates
integrations 295
project connections 1744
projects
connections 859
deleting 899
disconnecting 859
exporting 863
functional testing 127
importing items 863
new 1748
overview 858
Rational project associations 858
proxy examples 199
recording monitor 1801, 1807
script recording 1800
script templates
placeholders 1169
scripts
comments 1743, 1817
customizing 1167
empty 1746, 1747
helper superclass placeholders 1179
object map placeholders 1174
properties 1908, 1909
starting 1181
template customization 1166
template placeholders 1175
template properties 1774
verification point placeholders 1177
selecting logs 1831
toolbars 1910
tutorials
data-driven test creation 166
Visual Basic .NET scripting 147
verification points
comparisons 923, 1855
views
show checkouts 1775
show history 1776
functional tests
Java version support 1716
Linux 60
playback options 836, 837, 1796
playing back 111, 133

G

GEF 140, 1215, 1715
GEF applications
enabling 139
generic support 1121, 1714
getProperty command
description 1782, 1850
functional test scripts 885
getTestData method 1195, 1198
Google Chrome 784
browser support 785
enable 787
enable from Functional Test 787
enable logging 856
extension 789
add extension 788, 789
IBM Rational
Rational
Functional Tester
HCL
OneTest
UI
for Google Chrome
788, 789
web store 788, 789
port number 789, 856
prerequisites 785
web server 789, 856
graphs
customizing 1483
grid controls
getting test data (example) 1198
group verification point 936, 936

H

helper Superclasses
new 1749
hierarchy 1439
HTML
application test scripts 883
application tests 1543
cross-browser test scripts 884
enabling browsers 780
functional test logs 1497
functional test support 1715
standard properties 1548
HTML files
Java applet testing 1547
HTML format
exporting reports 1488
HTML wrapper 816
HTTP or HTTPs web server 208

I

IARM (test execution services) 1044
IBM
Rational
Functional Tester
HCL
OneTest
UI
licenses 304
icons
test nplayback monitor 1443
ICustomCode2 (test execution services) 1044
IDataArea (test execution services) 1044
IEngineInfo (test execution services) 1044
ILoopControl (test execution services) 1044
image
verification points
actions 1852
image verification point 935
image verification points
creating 108
import objects 1885
importing
reports
to .view file 1490
InputKeys 576
insert dynamic test objects 1838
Inspector tool
test object properties 1838
install method 203
install product packages 293
installation
response files 221, 221
silent 220
terminology 202
installation location 204
installation manager 204, 208, 209, 293
Installation Manager
overview 203
installing
launchpad program 212
post-installation tasks 225
preinstallation 212
installing packages
Installation Manager 203
integration
Rational Quality Manager
keywords 770
Internet Explorer
configuring for testing 1871
IP addresses
retrieving from virtual user 1053
IPDLogManager (test execution services) 1044
IScalar (test execution services) 1044
IStat (test execution services) 1044
IStatistics (test execution services) 1044
IStatisticsManager (test execution services) 1044
IStatTree (test execution services) 1044
iteration count
playback options 1882
ITestExecutionServices (test execution services) 1044
ITestInfo (test execution services) 1044
ITestLogManager (test execution services) 1044
IText (test execution services) 1044
ITime (test execution services) 1044
ITransaction (test execution services) 1044

IVirtualUserInfo (test execution services) 1044
ivory.properties 1559

J

jar 802
java 875
Java
  browser plug-in enablement 800
  functional tests 1716
  test enablement 778
  test environment configurations 797
  test execution services 1042
Java applets
  testing in HTML pages 1547
java code snippet 875
Java editor
  test scripts 1785
java method 875
java module 876
Java Runtime Environment
  enabling for testing 1785
  seaches 1819
  test configurations 797
  test enablement 778
java scripts 877
  simplified statements 865
Jazz
  project
    sharing 516
JVM heap size
  retrieving maximum 1062
JVM_Info class 1062

K

keyboard actions in tests
  playbacks 1204
keyboard shortcuts
  mapping 844, 845, 854
keyword view 500, 770
keywords 497, 497, 497
  associate scripts 498
  associating with manual tests 770
  viewing 770

L

launchpad 208
launchpad program 209
license expiration terms
  viewing 304
license types
  viewing 304
licenses
  managing 302
  testing 304
  viewing information on packages 304
Linux
  functional tests 60
lists
  data extraction in functional tests 1202
local application 814
local test computer 137, 137, 179
log 872, 1892
Log extension 1502
Logon 497
logs 1491, 1559
  Dojo 1502
  exporting test events 1494
  functional test formats 1497
  functional test scripts 892
  functional test selection 1831
  levels
    SAP performance tests 714
  managing test logs 1501
  messages in functional test scripts 1788, 1817
  overview 1493
  playback options for tests 836, 837, 1796
  preferences for functional tests 834, 1498, 1789
  renaming 1502
  test run messages 1179
loops
  controlling 1048

M

manage counters 1485
managing licenses
  overview 302
message 872
messages
  test logs 1179
  types
    recorder monitor preferences 844, 1809
methods
  CallScript
    parameters 1200
MFC controls 1720
Microsoft HTML applications
  testing 1543
migration
  custom Java code 1074
mobile and web ui data 596
mobile data 597
modify installations
  change features 228
  change language 228
  modify package wizard 228
modify packages 293
modify step target 597
modifying packages
  installation manager 203
monitor
  recording 1801
  recording toolbar 1807
mouse actions in tests
  playbacks 1204
Mozilla
  configuring for testing 1871

N

navigation actions 576
nested domain 1721
nested domains
  test application domain support 1896

O

object actions
  application controls 1890
object library 1883
  add object 1884
  export objects 1886
  import objects 1885
object maps
  updating 117
  viewing 113
object not found 871
object property configuration tool 1883
objects
  combining 1841
  displaying information 885
  options for adding to datasets 1830
  properties
    accessing 1182
    updates 1825, 1842
  selecting 1827, 1845
  substitutions 1842
  wait state settings 886
operating systems
  preferences 836, 1794
options 831, 1779
  Delays 839, 1794
  development environment 1900
  functional testing
    object additions to maps 1832
    preferences 828, 1769
  monitor
    playback preferences 840, 1795
  operating system
    preferences 836, 1794
  playback options for tests 836, 1796
  Recorder
    recorder preferences 843, 1810
  recorder monitor 844, 1809
  ScriptAssure
    recognition score preferences 840, 1797
  ScriptAssure(TM)-standard 841, 1798
  setting playback options in scripts 837
OutOfScopeException (test execution services) 1044

P

pages
  error handling 676
  title specifications in tests 680
  viewing test data 652, 660
  viewing test request data 660
palettes 140
parameters
  passing to the CallScript Method 1200
pause 871
pdf 1704
pdf form 1704
performance testing
  guidelines (SAP) 714
play back 497
playback 871, 1439, 1892
  environment
    restoring 1437
  options 1882
  test monitor icons 1443
playback interactive 836, 837, 1796
playbacks
  delays in scripts 894
post-installation tasks
  checklist 225
powerbuilder 1722
PowerBuilder 1217
preferences
  changing fonts 828
  functional testing 1799, 1915
    recording monitor 1792
    workbench 858, 1875
  log options for functional tests 834, 1498, 1789
  playback monitor 840, 1795
  recorder monitor 844, 1809
  test debug 1449
  test generation
    changing SAP preferences 722
  test logs 1500
preinstallation

install 210
prerequisites
recording 774, 778
PressKey 576
PrintArgs class 1054
problem identification
data correlation 672
problems 685
process
high level interaction 1080
model 1079
product package 208
programmatic screen scrapping
data verification 1006
projects
adding folders to 1745
connections
functional testing 1744
functional testing
connections 859
creating 858
importing an .rftjdtr file 863
new 1748
overview 858
items
exporting 863
importing 863, 1780
overwriting items 1830
properties 1892
objects
querying values 1182
test objects 1838
values
get property command 885
standard 1548
verification points
actions 1852
Visual Studio .NET 59
properties verification points
creating 108, 904, 1783, 1850
Properties view
editing simplified scripts 867
General page 1890
proxies
exporting items 1124
importing packages 1125
proxy 1091
.Net domain 1144
architecture 1079
best practices 1163
creating 1096
current support level 1093
debug 1119
debug log 1120
development 1090
environment 1092
enhancing recording behavior 1104
enhancing recording behavior with SubItems 1107
exceptions 1161
extending proxies 1096
hierarchy 1134
Java domain 1138
mapping to controls 1114
model 1083
sdk 1078
setting up projects 1092
TestData type 1156
proxy classes
creating 1126
proxy development
wizard driven development 1123
proxy examples 199
proxy items
exporting 1127
importing 1128
proxy packages
importing 1125
Proxy SDK wizard 1122

R

Rational Quality Manager 484, 497
adapter 365
configuring the adapter 358
passing arguments 493
passing execution variables 495
starting the adapter 362
testing shared assets 366
Rational Team Concert 514
defect tracking 356
recognition
test object 1443, 1542
test objects 1184
verification point data 917, 1865
recognition properties
objects
updates 1842
updates 1825
recognition scores
ScriptAssure preferences 840, 1797
ScriptAssure(TM) page-standard 841, 1798
warning thresholds 1441
record 498, 866
record selection order
datasets 953
playback options 1882
Record Toolbar
The Recording User Actions Toolbar 1802
recorder monitor
preferences 844, 844, 1809, 1809
Recorder page
recorder preferences 843, 1810
recording
SAP tests
batch input tests 719
inserting new recordings 719
overview 714
procedures 716
script 902
troubleshooting tips 899
recording monitor
functoinal testing 1801
toolbar 1807
Recording Monitor
messages
time stamping 1793
selecting text color
TextColorHelp 1792
toolbar 1802
references
test datasets 660
registry entries
reading in a functional test 1193
regression testing
automating 1437
regression tests
performing 115
regular expressions
evaluation 1810
using regular expressions 122
reports
.view format exports 1490
changing default 1485
customizing graphs 1483
filtering results 1481
HTML format exports 1488
repository
repository preferences 208
repository preferences 209
Request Time verification point (SAP) 730
required privileges 210
response codes
specifying expected 681
response files 221
creating 221
silent installation 220
response size
specifying 682
Response time breakdown
Response
breakdown 808
time 808
results
CSV format exports 1489
filtering 1481
rftjdtr file
generating 863
importing 863
roadmap 36, 47, 49, 53, 57
Runtime Loader 818

S

samples
code 1190
SAP
batch connection details 731
batch input transaction details 732
call details 729
configuration 714
connection details 725
get details 728
recording tests
batch input 719
inserting sequences 719
performance tests 716
screen details 726
set details 727
support
SAP GUI HTML 1724
SAP Support 1724
test details 724
test generation preferences 722
SAP client 805
SAP Support
enable SAP GUI 805
enable SAP server 806
schedules
command-line starts 1381
screen captures
playback failures 1450, 1542
screen shot 872
script 866
script arguments
passing from Rational Quality Manager 493
script editor 866
simplified script 1889
script execution
modifying options for 1180
Script Explorer 1813
Visual Studio .NET 59

script helper classes
adding to scripts 1829
functional test project properties 1908
overview 888
properties
XDE Tester 1770
superclasses
creating 889
SuperClasses
changing 887, 889
XDE Tester script properties 1909
script templates
customization 1166
customizing 1166, 1167
functional testing properties 1774
object map placeholders 1174
placeholders 1169
script helper super class
placeholders 1179
template placeholders 1175
verification point placeholders 1177
ScriptAssure
options 840, 1797
recognition properties 121
warning thresholds 1441
scripting framework 1179
scripting SDK 1179
scripts
assets 1834
coding manually 879
combined objects 1841
comments 1743, 1817
creating without recording 880
data-driving 938
dataset associations 954
debugging 1449
delays 1818
deleting 899
functional testing
comments 895
empty 1746, 1747
recording 1800
functional testing calls 891, 1739, 1816
helper classes 1829
importing 1780
modifying 1179
name renewal 1812
object substitutions 1842
recording 107, 127, 877
renaming 897, 1811
saving 898
starting 1181
stopping playback 1444
test development phases 1437
test object map associations 1737
timers 1818
versioning 305
sdk 1078
searches
filters 1835
test object maps
Quick Find 1800
security
overview mcmxvii
security settings 137
Selenium
adding to compound tests 734, 735
Appium 734
importing Appium Java projects 734
importing Selenium Java projects 734, 734
importing tests 734, 735
managing tests 733
modifying tests 736
running tests 736
viewing tests 736
server 805
Server Explorer
Visual Studio .NET 59
server request details 730
SetCookieFixedValue class 1056
setup tasks 774, 778
shared assets
testing 366
silent installation 220
response files 221
simplified script 866, 944
preference 856
verification points 934
simplified scripts 865
datasets 945
functional testing 865
simplify scripts
enabling 855
simulating non html actions 576
Solution Explorer
Visual Studio .NET 59
source control
element histories 1776
special keys 576
specific support 1121, 1714
split points
inserting during recording 571
start page
Visual Studio .NET 59
startApp command 1837
starting
installation manager 212
launchpad program 212
statistics
CSV format exports 1489
statistics sample interval
long performance test settings (SAP) 714
StatType (test execution services) 1044
strings
managing 1059
substitutions
data correlation 668
superclass
overview 888
support
eclipse 1711
switch to java 875, 876
syntax
changes 827

T

table cells
getting test data (example) 1198
values 1192
team concert 514
templates
customizing 1167
customizing for scripts 1166
functional tester script properties 1774
object map placeholders 1174
placeholders 1169
script helper super class
placeholders 1179
script placeholders 1175
verification placeholders 1177
terminal-based applications 1726
terminology
package 202
product installation 202
test application domain support 1703
test applications
starting 896
test assets
importing into Rational Quality Manager 365
test data
recorrelating 676
sources 665
test editor 866
test enablement
Eclipse platforms for functional testing 1756
test execution services
code execution counts 1055
custom counters 1064
determining where a test is running 1058
interfaces and classes 1044
migrating Java code 1074
overview 1042
printing input arguments to a file 1054
retrieving the maximum JVM heap size 1062
retrieving virtual user IP address 1053
running a program with a test 1062
setting and clearing cookies for virtual users 1056
statistics 1066
transactions 1066
using strings 1059
Test Flex application 134, 136, 137, 137, 175, 177, 179
test flow 36, 47
test lines
properties 1890
test objects
add dynamic test objects 1734
combining 1841
converting dynamic test objects 1734
data driven tests
actions 1781
data-driving 1833
deletions 1755, 1756
description customization 1757
descriptions 1836
displaying information 885
dynamic test objects
converting mapped objects 1734
filter searches 1751
getProperty command 1782, 1850
highlighting 1836
maps
adding objects 1823
color changes 826, 830
description customization 1757
importing 1780
new 1745, 1747
object additions 1832
object merges 1791
overview 1838
renaming 1811
script associations 1737
script renaming 897
search filters 1835
searches with Quick Find 1800
name renewal 1812, 1812

properties 1838
recognition 1852
ambiguous 1184, 1443, 1542
renaming 1811
search filters 1752, 1753
searching for SAP TestObjects 1210
searching for TestObjects 1206
selecing 1827, 1845
selecting for tests 1844
selections 1821
substitutions 1842
unregistering references to 1183
unused 1754
updating recognition properties for 1740
Tester 175
testing
application 36, 47
Eclipse application 53
Flex application 57
HTML application 51
Java application 49
SAP application 55
testing environment 134, 137, 175
TestManager
exporting to datasets 1780
functional test logs 1497
TestObject 1128
adding new TestObjects 1129
class diagram 1146
role 1128
tests
adding custom Java code 1042
annotating during recording 570
annotations
adding during recording 570
application types supported 1896
customizing 1044
data driven
objects 1833
data-driven
overview 937
data-driving actions 1781
dataset column associations 651
dataset references 648
datasets 658, 659, 659, 946, 946
SAP options 724
development phases 1437
editing
Web UI
573
elements
source control 1776
extending
controlling loops 1048
folders
renaming 1811
generating
preferences (SAP) 722
log preferences 834, 1498, 1789
logs
exporting 1494
functional test formats 1497
long performance test settings (SAP) 714
managing 1501
messages 1179
overview 1493
manual data correlation 667
migrating custom Java code 1074
new
adding 1898
playback environments 1437
recording in SAP 716
recording SAP batch input 719
SAP
batch connection details 731
batch input transaction details 732
call details 729
get details 728
inserting new recordings 719
screen details 726
server request details 730
set details 727
scripts
functional test messages 892
literal value substitutions 1749
literal values substitutions 1899
log messages in functional tests 1788, 1817
splitting during recording 571
test execution services
code execution counts 1055
custom counters 1064
determining where a test is running 1058
printing input arguments to a file 1054
retrieving the maximum JVM heap size 1062
retrieving virtual user IP address 1053
running a program with a test 1062
setting and clearing cookies for virtual users 1056
statistics 1066
transactions 1066
using strings 1059
types performed 1827, 1845
types to perform 1844
viewing errors 1494
text
functional test logs 1497
text editor
Visual Studio .NET 59
think time
SAP 729
get details 728
set details 727
time offset
correcting 1486
time stamps
recording monitor messages 1793
timers
adding to scripts 1818
test scripts 893
TN3270 host 962
TN3270E 962
toolbars
Functional Test 1777
functional testing 1910
Visual Studio .NET 59
Object Map 1838
recording 1802
TransactionException
test execution services 1044
tree controls
getting data from 1195
troubleshoot 685, 1507, 1559
troubleshooting
data correlation 675
recording tips 899
trust designations 136, 178
tutorials
data-driven test creation 166
data-driven tests 168, 169, 171, 172
functional testing
Visual Basic .NET scripting 147

U

UI controls
data extraction in functional tests 1202
unexpected active windows
handling 1188
update application visual 874
update packages 293
updating packages
Installation Manager 203
UrbanCode Deploy 529
UrbanCode Deploy plugin 529
user interface
font changes 828
user privileges 201

V

value class 1152
value manager 1152
values
table cells 1192
variable columns
datasets
insertions 1735
variables
names
verification points 1844, 1851
verification point 936
data 911
data type 911
object selection 911
verification points
adding
while recording 1820, 1847
without recording 1820, 1847
adding manual and dynamic 1186
color changes 826, 830
comparator 115
creating 108, 130
data 855, 907
data selections 1852
data verification points 1784, 1848
dataset references 951
editing 917, 1865
expected response codes 681
functional testing
comparisons 923, 933, 1855
image 1849
creating 914
literal value conversions 1751
page title test specifications 680
performance testing
global enablement 680
properties 904
properties verification points 1783, 1850
property 855
recording 1827, 1845
recording for tests 1844
renaming 1811
response size specifications 682
SAP
enabling 728
request time 730
screen title 726
syntax changes 827
variable names 1844, 1851
viewing 113

view 497
view file
exporting reports 1490
importing reports 1490
views
bookmarks 1738
compiler messages 1838
functional testing
console 1744
virtual users
counting code runs 1055
datasets 648
retrieving IP addresses 1053
setting and clearing cookies 1056
visual basic 1725
Visual Basic .NET
functional testing
command-line scripting
interface 1893
visual object maps 855
visual objects
simplified scripts 865
Visual Studio .NET
class view 59
support 1717

W

wait for a control 871
wait state
object settings 886
waitForExistence command 1784, 1851
warnings
displaying views 1838
web application 814
web browsers
enabling 780, 799, 1738, 1758
hard disk searches 1819
Web browsers
configuring 798
Web UI
test editor overview 573
Windows
support 1717, 1720
Windows registry
reading in a functional test 1193
WinForm 1721
wizard driven development
proxies 1124
wizards
creating proxies 1123
workbench
functional testing
preferences 858, 1875
Workbench Preferences 857, 1874

X

XDE Tester
script properties 1909