Development Testing Platform Engines for Java User's Guide

Version 10.3

Parasoft Corporation
101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Phone: (888) 305-0041
Fax: (626) 305-9048
E-mail: info@parasoft.com
URL: www.parasoft.com

PARASOFT END USER LICENSE AGREEMENT

PLEASE READ THIS END USER LICENSE AGREEMENT ("AGREEMENT") CAREFULLY BEFORE USING THE SOFTWARE. PARASOFT CORPORATION ("PARASOFT") IS WILLING TO LICENSE THE SOFTWARE TO YOU, AS AN INDIVIDUAL OR COMPANY THAT WILL BE USING THE SOFTWARE ("YOU" OR "YOUR") ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS OF THIS AGREEMENT. THIS IS A LEGALLY ENFORCEABLE CONTRACT BETWEEN YOU AND PARASOFT. BY CLICKING THE "ACCEPT" OR "YES" BUTTON, OR OTHERWISE INDICATING ASSENT ELECTRONICALLY, OR BY INSTALLING THE SOFTWARE, YOU AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT AND ALSO AGREE THAT IS IT ENFORCEABLE LIKE ANY WRITTEN AND NEGOTIATED AGREEMENT SIGNED BY YOU. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, CLICK THE "I DO NOT ACCEPT" OR "NO" BUTTON AND MAKE NO FURTHER USE OF THE SOFTWARE.

1. DEFINITIONS

1.1. **"Community Edition"** means a limited version of certain Software available with a no cost license. You can execute only one Instance of a Community Edition on a single machine. You shall provide Parasoft with a valid email address at the time of installation, and such email address cannot be used by any other individual to register a Community Edition. You may not transfer the Community Edition to another machine without prior written approval from Parasoft. You may not tamper or attempt to bypass any of the installation steps for a Community Edition, and Parasoft shall terminate your right to a Community Edition in the event that you do so. Notwithstanding any other provisions herein, Parasoft (a) does not provide Maintenance for Community Editions; (b) provides no warranty for Community Editions; (c) provides no indemnification for Community Editions; and (d) accepts no liability for Community Editions.

1.2. **"Concurrent User"** means a person that has accessed the Software at any given point in time, either directly or through an application.

1.3. **"Instance"** means a single occurrence of initialization or execution of software on one machine. You are prohibited from using more than one Instance on the same machine at the same time.

1.4. **"Licensed Capacity"** means the capacity-based license pricing metrics, including, without limitation, Concurrent Users, Node Locked machines, Instances, and Community Editions.

1.5. **"Maintenance"** means the maintenance and technical support services for the Software identified in the Order Instrument and provided by Parasoft pursuant to this Agreement.

1.6. **"Node Locked"** means a license for a single machine that has been authorized to run a single Instance of the licensed Software. A Node Locked license requires that users are physically present and not accessing the machine and using the Software from a remote location.

1.7. **"Software"** means Parasoft's software products, in object code form, that are commercially available at the time of Your order and identified on the Order Instrument, and any modifications, corrections and updates provided by Parasoft in connection with Maintenance.

1.8. **"Territory"** means the country or countries in which You have a license to use the Software, as specified in Your order for the Software; or, if no Territory is specified, the country from which Your order has been issued.

1.9. **"User Documentation"** means the user's guide, installation guides, and/or on-line documentation applicable to the Software. User Documentation does not include marketing materials or responses to requests for proposals.

2. GRANT OF LICENSE AND USE OF SOFTWARE

2.1. **License Grant.** Subject to the terms and conditions of this Agreement, Parasoft grants to You a non-exclusive license to use the Software within the Territory, in accordance with the User Documentation and in compliance with the authorized Licensed Capacity. This license may be perpetual or for a limited duration term, as stated in (a) an executed agreement between You and Parasoft; (b) a sales quotation from Parasoft; (c) a purchase order that You issue to Parasoft; or (d) the online ordering process found on Parasoft's website or an authorized third party's website. You acknowledge and agree that this Agreement only grants a license to the Software as set forth herein and does not constitute a sale of the Software by Parasoft. You have no right to resell the Software, whether by contract or by operation of applicable copyright law.

2.2. **Usage Rights.** You may only use the Software and/or the User Documentation for Your internal business operations and to process Your data. You shall not (a) permit any third parties or non-licensed entities to use the Software or the User Documentation; (b) process or permit to be processed any data that is not Your data; (c) use the Software in the operation of a service bureau; (d) sublicense, rent, or lease the Software or the User Documentation to a third party; or (e) perform, publish, or release to any third parties any benchmarks or other comparisons regarding the Software or User Documentation. You shall not make simultaneous use of the Software on multiple, partitioned, virtual, or cloud hosted computers without first procuring an appropriate number of licenses from Parasoft. You shall not bypass or attempt to bypass any licensing controls either contained within the Software or imposed by Parasoft. You shall not permit a third party outsourcer to use the Software to process data on Your behalf without Parasoft's prior written consent.

2.3. **License Keys.** You acknowledge that the Software contains one or more license keys that will enable the functionality of the Software and third party software embedded in or distributed with the Software. You may only access and use the Software with license keys issued by Parasoft, and shall not attempt to modify, tamper with, reverse engineer, reverse compile, or disassemble any license key. If Parasoft issues a new license key for the Software, You shall not use the previous license key to enable the Software. If a particular license is then currently on Maintenance, You may transfer such license to a different machine and request a new license key from Parasoft.

2.4. **Archival Copies.** You may make one copy of the Software for back-up and archival purposes only. You may make a reasonable number of copies of the User Documentation for Your internal use. All copies of Software and User Documentation must include all copyright and similar proprietary notices appearing on or in the originals. Copies of the Software may be stored offsite provided that all persons having access to the Software are subject to Your obligations under this Agreement and You take reasonable precautions to ensure compliance with these obligations. Parasoft reserves the right to revoke permission to reproduce copyrighted and proprietary material if Parasoft reasonably believes that You have failed to comply with its obligations hereunder.

2.5. **Licensed Capacity.** Parasoft licenses Software based on Licensed Capacity for different types of usage, including, without limitation, Concurrent Users, Node Locked machines, and Community Editions. A Concurrent User license allows multiple Concurrent Users to share access to and use the Software, provided that the number of Concurrent Users accessing the Software at any time does not exceed the total number of licensed Concurrent Users. A Node Locked license allows a single specified machine to run a single Instance of the Software. If an application accessing the Software is a multiplexing, database, or web portal application that permits users of such application to access the Software or data processed by the Software, a separate Concurrent User license will be required for each Concurrent User of such application. Regardless of usage type, You shall immediately notify Parasoft in writing of any increase in use beyond the Licensed Capacity. You must obtain a license for any increase in

Licensed Capacity, and You agree to pay to Parasoft additional Software license fees, which will be based on Parasoft's then-current list price.

2.6. **Third Party Terms.** You acknowledge that software provided by third party vendors ("Third Party Software") may be embedded in or delivered with the Software. The terms of this Agreement and any other terms that Parasoft may specify will apply to such Third Party Software, and the Third Party Software vendors will be deemed third party beneficiaries under this Agreement. You may only use the Third Party Software with the Software. You may not use the Third Party Software on a stand-alone basis or use or integrate it with any other software or device.

2.7. **Evaluation License.** This Section 2.7 applies if Parasoft has provided the Software to You for evaluation purposes. Parasoft grants to You a thirty (30) day, limited license solely for the purpose of internal evaluation. You are strictly prohibited from using the Software for any production purpose or any purpose other than the sole purpose of determining whether to purchase a commercial license for the Software that You are evaluating. Parasoft is not obligated to provide maintenance or support for the evaluation Software. YOU ACKNOWLEDGE THAT SOFTWARE PROVIDED FOR EVALUATION MAY (A) HAVE LIMITED FEATURES; (B) FUNCTION FOR A LIMITED PERIOD OF TIME; OR (C) HAVE OTHER LIMITATIONS NOT CONTAINED IN A COMMERCIAL VERSION OF THE SOFTWARE. NOTWITHSTANDING ANYTHING TO THE CONTRARY IN THIS AGREEMENT, PARASOFT IS PROVIDING THE EVALUATION SOFTWARE TO YOU "AS IS", AND PARASOFT DISCLAIMS ANY AND ALL WARRANTIES (INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND STATUTORY WARRANTIES OF NON-INFRINGEMENT), LIABILITIES, AND INDEMNIFICATION OBLIGATIONS OF ANY KIND. In the event of any conflict between this Section 2.7 and any other provision of this Agreement, this Section 2.7 will prevail and supersede such other provision with respect to Software licensed to You for evaluation purposes.

2.8. **Education License.** If You are an educational or academic institution and are receiving a discount from Parasoft, You may use the Software solely for education or academic purposes and You may not use the Software for any commercial purpose. Parasoft may require that You provide proof of your status as an educational or academic institution.

2.9. **Audit.** You shall maintain accurate business records relating to its use and deployment of the Software. Parasoft shall have the right, not more than once every twelve (12) months and upon ten (10) business days prior written notice, to verify Your compliance with its obligations under this Agreement by auditing Your business records and Your use and deployment of the Software within Your information technology systems. Parasoft and/or a public accounting firm selected by Parasoft shall perform the audit during Your regular business hours and comply with Your reasonable safety and security policies and procedures. Any agreement You may require the public accounting firm to execute shall not prevent disclosure of the audit results to Parasoft. You shall reasonably cooperate and assist with such audit. You shall, upon demand, pay to Parasoft all license and Maintenance fees for any unauthorized deployments and/or excess usage of Software products disclosed by the audit. License fees for such unauthorized deployments and/or excess usage shall be invoiced to and paid by You at Parasoft's then-current list price, and applicable Maintenance fees shall be applied retroactively to the entire period of the unauthorized and/or excess usage. Parasoft shall be responsible for its own costs and expenses in conducting the audit, unless the audit indicates that You have exceeded its Licensed Capacity or otherwise exceeds its license restrictions, such that the then-current list price of non-compliant Software deployment exceeds five percent (5%) of the total then-current list price of the Software actually licensed by You, in which event You shall, upon demand, reimburse Parasoft for all reasonable costs and expenses of the audit.

3. **TITLE**. Parasoft retains all right, title and interest in and to the Software and User Documentation and all copies, improvements, enhancements, modifications and derivative works of the Software and User Documentation, including, without limitation, all patent, copyright, trade

secret, trademarks and other intellectual property rights. You agrees that it shall not, and shall not authorize others to, copy (except as expressly permitted herein), make modifications to, translate, disassemble, decompile, reverse engineer, otherwise decode or alter, or create derivative works based on the Software or User Documentation. Except as otherwise provided, Parasoft grants no express or implied rights under this license to any of Parasoft's patents, copyrights, trade secrets, trademarks, or other intellectual property rights..

4. TERMINATION

4.1. **Default; Bankruptcy**. Parasoft may terminate this Agreement if (a) You fail to pay any amount when due under any order You have placed with Parasoft and do not cure such non-payment within ten (10) days of receipt of written notice of non-payment; (b) You materially breach this Agreement and do not cure such breach within thirty (30) days of receipt of written notice of such breach; (c) subject to provisions of applicable bankruptcy and insolvency laws, You become the subject of any involuntary proceeding relating to insolvency and such petition or proceeding is not dismissed within sixty (60) days of filing; or (d) You become the subject of any voluntary or involuntary petition pursuant to applicable bankruptcy or insolvency laws, or request for receivership, liquidation, or composition for the benefit of creditors and such petition, request or proceeding is not dismissed within sixty (60) days of filing.

4.2. **Effect of Termination**. Upon termination of this Agreement, You shall immediately discontinue use of, and uninstall and destroy all copies of, all Software. Within ten (10) days following termination, You shall certify to Parasoft in a writing signed by an officer of Yours that all Software has been uninstalled from Your computer systems and destroyed.

5. LIMITED WARRANTY

5.1. **Performance Warranty**. Parasoft warrants that the Software, as delivered by Parasoft and when used in accordance with the User Documentation and the terms of this Agreement, will substantially perform in accordance with the User Documentation for a period of ninety (90) days from the date of initial delivery of the Software. If the Software does not operate as warranted and You have provided written notice of the non-conformity to Parasoft within the ninety (90) day warranty period, Parasoft shall at its option (a) repair the Software; (b) replace the Software with software of substantially the same functionality; or (c) terminate the license for the nonconforming Software and refund the applicable license fees received by Parasoft for the nonconforming Software. The foregoing warranty specifically excludes defects in or non-conformance of the Software resulting from (a) use of the Software in a manner not in accordance with the User Documentation; (b) modifications or enhancements to the Software made by or on behalf of You; (c) combining the Software with products, software, or devices not provided by Parasoft; or (d) computer hardware malfunctions, unauthorized repair, accident, or abuse.

5.2. **Disclaimers**. THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, AND PARASOFT EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND STATUTORY WARRANTIES OF NON-INFRINGEMENT. PARASOFT DOES NOT WARRANT THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT USE OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE. THE REMEDIES SET FORTH IN THIS SECTION 5 ARE YOUR SOLE AND EXCLUSIVE REMEDIES AND PARASOFT'S SOLE AND EXCLUSIVE LIABILITY REGARDING FAILURE OF ANY SOFTWARE TO FUNCTION OR PERFORM AS WARRANTED IN THIS SECTION 5.

6. INDEMNIFICATION

6.1. **Infringement**. Parasoft shall defend any claim against You that the Software infringes any intellectual property right of a third party, provided that the third party is located in a country that is a signatory to the Berne Convention, and shall indemnify You against any and all damages finally awarded against You by a court of final appeal, or agreed to in settlement by Para-

soft and attributable to such claim, so long as You (a) provide Parasoft prompt written notice of the claim; (b) provide Parasoft all reasonable assistance and information to enable Parasoft to perform its duties under this Section 6; (c) allow Parasoft sole control of the defense and all related settlement negotiations; and (d) have not compromised or settled such claim. If the Software is found to infringe, or if Parasoft determines in its sole opinion that it is likely to be found to infringe, then Parasoft may, at its option (a) obtain for You the right to continue to use the Software; (b) modify the Software to be non-infringing or replace it with a non-infringing functional equivalent, in which case You shall stop using any infringing version of the Software; or (c) terminate Your rights and Parasoft's obligations under this Agreement with respect to such Software and refund to You the unamortized portion of the Software license fee paid for the Software based on a five year straight-line depreciation schedule commencing on the date of delivery of the Software. The foregoing indemnity will not apply to any infringement resulting from (a) use of the Software in a manner not in accordance with the User Documentation; (b) modifications or enhancements to the Software made by or on behalf of You; (c) combination, use, or operation of the Software with products not provided by Parasoft; or (d) use of an allegedly infringing version of the Software if the alleged infringement could be avoided by the use of a different version of the Software made available to You.

6.2. Disclaimers. THIS SECTION 6 STATES YOUR SOLE AND EXCLUSIVE REMEDY AND PARASOFT'S SOLE AND EXCLUSIVE LIABILITY REGARDING INFRINGEMENT OR MISAPPROPRIATION OF ANY INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY.

7. **LIMITATION OF LIABILITY**. IIN NO EVENT WILL PARASOFT OR ITS THIRD PARTY VENDORS BE LIABLE TO YOU OR ANY OTHER PARTY FOR (A) ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR (B) LOSS OF DATA, LOSS OF PROFITS, BUSINESS INTERRUPTION, OR SIMILAR DAMAGES OR LOSS, EVEN IF PARASOFT AND ITS THIRD PARTY VENDORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. EXCEPT AS LIMITED BY APPLICABLE LAW AND EXCLUDING PARASOFT'S LIABILITY TO YOU UNDER SECTION 6 (INDEMNIFICATION), AND REGARDLESS OF THE BASIS FOR YOUR CLAIM, PARASOFT'S MAXIMUM LIABILITY UNDER THIS AGREEMENT WILL BE LIMITED TO THE LICENSE OR MAINTENANCE FEES PAID FOR THE SOFTWARE OR MAINTENANCE GIVING RISE TO THE CLAIM. THE FOREGOING LIMITATIONS WILL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.
8. **CONFIDENTIAL INFORMATION**. For purposes of this Agreement, "Confidential Information" will include trade secrets contained within the Software and User Documentation, the terms and pricing of the Software and Maintenance (including any pricing proposals), and such other information (a) identified by either party as confidential at the time of disclosure or (b) that a reasonable person would consider confidential due to its nature and circumstances of disclosure ("Confidential Information"). Confidential Information will not include information that (a) is or becomes a part of the public domain through no act or omission of the receiving party; (b) was in the receiving party's lawful possession prior to receiving it from the disclosing party; (c) is lawfully disclosed to the receiving party by a third party without restriction on disclosure; or (d) is independently developed by the receiving party without breaching this Agreement. Each party agrees to maintain all Confidential Information in confidence and not disclose any Confidential Information to a third party or use the Confidential Information except as permitted under this Agreement. Each party shall take all reasonable precautions necessary to ensure that the Confidential Information is not disclosed by such party or its employees, agents or authorized users to any third party. Each party agrees to immediately notify the other party of any unauthorized access to or disclosure of the Confidential Information. The receiving party agrees that any breach of this Section 8 may cause irreparable harm to the disclosing party, and such disclosing party shall be entitled to seek equitable relief in addition to all other remedies provided by this Agreement or available at law.
9. **MAINTENANCE**

9.1. **Maintenance Period**. If You have purchased a perpetual license, You are required to purchase first year Maintenance with the Software, and the Maintenance period will commence upon the initial delivery of the Software and continue for a period of one year. If You have purchased a term license, Maintenance during the term is included at no additional charge. The Maintenance period, at Your option, may be renewed pursuant to subsequent orders. Prior to such renewal, Parasoft may, upon ten (10) business days written notice, require You to provide a report on Your use and deployment of the Software. Such report will be certified by an officer of Yours and will specify, with respect to Your Software: (a) the type and amount of Licensed Capacity; (b) the version; and (c) the Parasoft license serial number. Parasoft shall issue an annual renewal notice to You at least ninety (90) days prior to the expiration of the then-current Maintenance period. Maintenance fees will be based on the then-current list price and are subject to change without notice.

9.2. **Support Coordinators**. Maintenance will consist of support services provided by Parasoft to one designated support coordinator of Yours (and one backup coordinator) per Your location, by telephone, email, and website. Support is available during normal business hours in the applicable location within the Territory, Monday through Friday, excluding nationally observed holidays.

9.3. **Additional Licensed Capacity**. Additional Licensed Capacity. In the event that You purchases additional Licensed Capacity for the Software prior to the annual anniversary date of the Maintenance period, You agree to pay applicable Maintenance fees based on Parasoft's then-current Maintenance rates. Maintenance fees will apply from the effective date of such additional Licensed Capacity and continue for a period of one year thereafter, unless otherwise agreed to in writing by the parties, so that Maintenance for Your previously acquired Software and added Licensed Capacity is coterminous.

9.4. **New Releases**. During any period in which You are current on Maintenance, Parasoft shall provide You with any new release of the Software, which may include generally available error corrections, modifications, maintenance patch releases, enhancements (unless priced separately by Parasoft and generally not included with new licenses for the Software at that time), and the revised User Documentation, if applicable. Notwithstanding the foregoing, stand-alone error corrections that are not part of a new release will not be independently supported but will be incorporated into the next release of the Software. If You install a new release of the Software, You may continue to use the previous version of the Software for up to ninety (90) days in order to assist You in the transition to the new release. Once You complete its transition to the new release of the Software, You must discontinue use of the previous version of the Software.

9.5. **Supported Releases**. Parasoft shall continue to support the immediately preceding release of the Software for a period of twelve (12) months following the discontinuance of such Software or the date on which the new release becomes generally available, provided that You have paid applicable Maintenance fees and incorporated all Maintenance patch releases issued by Parasoft for the release of the Software.

9.6. **Reinstatement of Maintenance**. If You allow Maintenance to expire, You may, at a later date, renew Maintenance by paying the following: (a) if You have installed the current release of the Software but have failed to pay the applicable renewal fee on or before the ninetieth (90th) day following expiration of the Maintenance period, annual Maintenance fees at Parasoft's then-current rates, plus Parasoft's then-current reinstatement fee; or (b) if You have not installed the current release of the Software or have failed to pay the applicable renewal fee by the ninetieth (90th) day following expiration of the Maintenance period, annual Maintenance fees at Parasoft's then-current rates, plus Parasoft's then-current license update fee for the current release of the Software.

10. GENERAL

10.1. **Independent Contractors**. The parties acknowledge and agree that each is an independent contractor. This Agreement will not be construed to create a partnership, joint venture or agency relationship between the parties.

10.2. **Entire Agreement**. The terms and conditions of this Agreement apply to all Software licensed, all User Documentation provided, and all Maintenance purchased hereunder. This Agreement will supersede any different, inconsistent or preprinted terms and conditions in any order form of Yours, purchase order or other ordering document.

10.3. **Assignment**. You have no right to assign, sublicense, pledge, or otherwise transfer any of Your rights in and to the Software, User Documentation or this Agreement, in whole or in part (collectively, an "Assignment"), without Parasoft's prior written consent, and any Assignment without such consent shall be null and void. Any change in control of Your organization or entity, whether by merger, share purchase, asset sale, or otherwise, will be deemed an Assignment subject to the terms of this Section 10.3.

10.4. **Force Majeure**. No failure, delay or default in performance of any obligation of a party to this Agreement, except payment of license fees due hereunder, will constitute an event of default or breach of the Agreement to the extent that such failure to perform, delay or default arises out of a cause, existing or future, that is beyond the reasonable control of such party, including, without limitation, action or inaction of a governmental agency, civil or military authority, fire, strike, lockout or other labor dispute, inability to obtain labor or materials on time, flood, war, riot, theft, earthquake or other natural disaster ("Force Majeure Event"). The party affected by such Force Majeure Event shall take all reasonable actions to minimize the consequences of any Force Majeure Event.

10.5. **Severability**. If any provision of this Agreement is held to be illegal or otherwise unenforceable by a court of competent jurisdiction, that provision will be severed and the remainder of the Agreement will remain in full force and effect.

10.6. **Waiver**. The waiver of any right or election of any remedy in one instance will not affect any rights or remedies in another instance. A waiver will be effective only if made in writing and signed by an authorized representative of the applicable party.

10.7. **Notices**. All notices required by this Agreement will be in writing, addressed to the party to be notified and deemed to have been effectively given and received (a) on the fifth business day following deposit in the mail, if sent by first class mail, postage prepaid; (b) upon receipt, if sent by registered or certified U.S. mail, postage prepaid, with return receipt requested; (c) upon transmission, if sent by facsimile and confirmation of transmission is produced by the sending machine and a copy of such facsimile is promptly sent by another means specified in this Section 10.7; or (d) upon delivery, if delivered personally or sent by express courier service and receipt is confirmed by the recipient. Notices will be addressed to the parties based on the address stated in the applicable order, to the attention of the Legal Department. A change of address for notice purposes may be made pursuant to the procedures set forth above.

10.8. **Export Restrictions**. You acknowledge that the Software and certain Confidential Information (collectively "Technical Data") are subject to United States export controls under the U. S. Export Administration Act, including the Export Administration Regulations, 15 C.F.R. Parts 730 et seq. (collectively, "Export Control Laws"). Each party agrees to comply with all requirements of the Export Control Laws with respect to the Technical Data. Without limiting the foregoing, You shall not (a) export, re-export, divert or transfer any such Technical Data, or any direct product thereof, to any destination, company, or person restricted or prohibited by Export Control Laws; (b) disclose any such Technical Data to any national of any country when such disclosure is restricted or prohibited by the Export Control Laws; or (c) export or re-export the Technical Data, directly or indirectly, for nuclear, missile, or chemical/biological weaponry end uses prohibited by the Export Control Laws.

10.9. **U. S. Government Rights**. The Software and User Documentation are deemed to be "commercial computer software" and "commercial computer software documentation" as defined in FAR Section 12.212 and DFARS Section 227.7202, as applicable. Any use, modification, reproduction, release, performance, display, or disclosure of the Software and User Documentation by the United States government will be solely in accordance with the terms of this Agreement.

10.10. **Choice of Law; Jurisdiction**. This Agreement is governed by and construed in accordance with the laws of the State of California, U. S. A., exclusive of any provisions of the United Nations Convention on Contracts for the International Sale of Goods, including any amendments thereto, and without regard to principles of conflicts of law. Any suits concerning this Agreement will be brought in the federal courts for the Central District of California or the state courts in Los Angeles County, California. The parties expressly agree that the Uniform Computer Information Transactions Act, as adopted or amended from time to time, will not apply to this Agreement or the Software and Maintenance provided hereunder.

10.11. **Amendment**. This Agreement may only be modified by a written document signed by an authorized representative of Parasoft and by You.

10.12. **Survival**. Any terms of this Agreement which by their nature extend beyond the termination or expiration of this Agreement will remain in effect. Such terms will include, without limitation, all provisions herein relating to limitation of liability, title and ownership of Software, and all general provisions.

Parasoft Corporation

101 East Huntington Drive, 2nd Floor

Monrovia, CA 91016 USA

+1 (626) 256-3680

+1 (888) 305-0041 (USA only)

+1 (626) 256-9048 (Fax)

info@parasoft.com

http://www.parasoft.com

Printed in the U.S.A, May 16, 2017

Table of Contents

Reporting

Unit Test Connector

Code Coverage Engine

Customizing DTP Engines for Java

Integrations

Getting Help

Third-Party Content

Introduction

Parasoft Development Testing Platform (DTP) Engines for Java are integrated solutions for automating a broad range of best practices to improve productivity and software quality. DTP Engines are a component of the Parasoft Development Testing Platform family of software quality solutions.Please read the following guide for additional information about how DTP Engines integrate into Parasoft's Development Testing ecosystem:

The Parasoft Development Testing Solution (PDF)

This documentation provides information on how to use the following engines:

Static Analysis Engine (SAE)

SAE enforces your coding policy with proven quality practices, such as static analysis and flow analysis, to ensure that your Java applications function as expected. See "Static Analysis Engine", page 10.

Unit Test Connector (UTC)

UTC allows you to run unit tests from open format tools, and report results to Development Testing Platform (DTP) Server. See "Unit Test Connector", page 43.

Code Coverage Engine (CCE)

CCE collects coverage information during a run of the executable and generates reports that can be sent to DTP Server. See "Code Coverage Engine", page 47.

Getting Started

This chapter will help you verify that your system meets the requirements for using DTP Engines, as well as help you configure DTP Engines so you can quickly start analyzing code.

System Requirements

Windows 32-bit

- Windows 7, Windows 8
- 4GB memory minimum*
- 2GHz or faster processor (x86-compatible), multi-CPU configuration recommended

Windows 64-bit

- Windows 7 (x64), Windows 8 (x64), Windows 10, Windows 2008 Server (x64), Windows Server 2012
- 4GB memory minimum, 8GB recommended*
- 2GHz or faster processor (x86_64-compatible), multi-CPU configuration recommended

Linux 32-bit

- Linux kernel 2.6 (or newer) with glibc 2.9 (or newer)
- 4GB memory minimum*
- 2GHz or faster processor (x86-compatible), multi-CPU configuration recommended

Linux 64-bit

- Linux kernel 2.6 (or newer) with glibc 2.12 (or newer)
- 4GB memory minimum, 8GB recommended*
- 2GHz or faster processor (x86_64-compatible), multi-CPU configuration recommended

Mac OS X 64-bit

- OS X 10.10 Yosemite, OS X 10.11 El Capitan
- 4GB memory minimum, 8GB recommended*
- 2GHz or faster processor (x86_64-compatible), multi-CPU configuration recommended

We recommend using Java Runtime Environment provided by Oracle Corporation in your environment. Other free and open source implementations may cause issues with the proper functioning of DTP Engines for Java.

*DTP Engines for Java may allocate up to 1GB RAM on 32-bit machines or up to 2GB RAM on 64-bit machines for the Java Virtual Machine process. You can change memory allocation for the JVM process in the `[INSTALL_DIR]/etc/jtestcli.jvm` configuration file (`-Xmx` option).

Installing DTP Engines

1. Unpack the installation package in any directory.

2. Make sure the environment contains the JAVA_HOME variable.
3. [Optional] Add the Static Analysis Engine directory path to $PATH or symlink `jtestcli.exe` (`jtestcli` in linux) into directory in $PATH
4. [Optional] Add the $JTEST_HOME environment variable pointing to the Static Analysis Engine directory path.
5. [Optional] Set the following properties in your Maven settings.xml file: `pluginRepository`, `groupId`, `jtest.home`. To view detailed instructions on setting these properties:
    a. Open `[INSTALL_DIR]manuals/plugins-manual.html`
    b. In the JTEST MAVEN PLUGIN menu, choose **Usage> Initial Setup**
6. [Optional] Deploy `[INSTALL_DIR]/integration/ant/jtest-ant-plugin.jar` into the `$ANT_HOME/lib` directory. To view detailed instructions on deploying the Ant plug-in jar:
    a. Open `[INSTALL_DIR]manuals/plugins-manual.html`
    b. In the JTEST ANT PLUGIN menu, choose **Usage> Initial Setup**

JVM, Framework, and Application Setup

The following table describes additional configuration files for setting up DTP Engines.

FIle	Description	Directory
`jtestcli.properties`	Contains the default settings for the Static Analysis Engine properties.	[INSTALL_DIR]
`jtestcli.jvm`	Contains JVM arguments that the `jtest-cli.exe` (`jtestcli`) executable will use when starting Java processes.	[INSTALL_DIR]/etc
`framework.properties`	Contains properties that are passed to the launched Felix OSGI framework. There is typically no need to edit this file.	[INSTALL_DIR]/etc
`formatting.properties`	Contains formatting rules for the default Static Analysis Engine properties.	[INSTALL_DIR]/etc
`logging.xml`	Logger configuration file; outputs a silent console by default and warn level on the `jtest.log` file.	[INSTALL_DIR]/etc
`logging.console.debug.xml`	Logger configuration with debug level on console output	[INSTALL_DIR]/etc

If you are using Azure or AWS services, you need to configure the `cloudvm` option in the .properties configuration file. You can set the option to one of the following values:

- `azure` - Enables integration with Azure
- `aws` - Enables integration with AWS
- `true` - Enables integration with the automatically detected cloud computing platform
- `false` - Disables integration (the default value)

If you set the value to `false` or if the option is not configured, integration with Azure or AWS is disabled.

Setting the License

DTP Engines can run on either a local or a network license. There are two types of network licenses:

- `dtp`: This license is stored in DTP. Your DTP license limits analysis to the number of files specified in your licensing agreement. This is the default type when `license.use_network` is set to `true`.
- `ls`: This is a "floating" or "machine-locked" license that limits usage to a specified number of machines. This type of license is stored in DTP in License Server.

Network licenses are also available in three editions that determine what functionality is available:

- `desktop_edition`: Functionality is optimized for desktop usage.
- `server_edition`: Functionality configured for high performance usage in server command line mode.
- `custom_edition`: functionality can be customized.

Local License

In the .properties configuration file:

1. Set the `jtest.license.use_network` property to `false`
2. Set the `jtest.license.local.password` property with your password

Obtaining the Machine ID

If you are using a local license, you will need your machine ID to request a license from Parasoft. Run the following command from a command line window to obtain your machine ID:

```
jtestcli -machineID
```

Network License

In the .properties configuration file:

1. Set the `jtest.license.use_network` property to `true`
2. Set the `jtest.license.network.type`
3. Set the `jtest.license.network.edition`

Connecting to DTP Server

Connecting to DTP Server is required for licensing, as well as extending other team-working capabilities, such as:

- Reporting analysis to a centralized database (see “Sending Results to Development Testing Platform (DTP) Server”, page 41)
- Sharing test configurations
- Sharing static analysis rules

Modify the following settings in the `[INSTALL_DIR]\jtestcli.properties` file to configure the connection to DTP Server.

```
dtp.server=[SERVER]
dtp.port=[PORT]
dtp.user=[USER]
dtp.password=[PASSWORD]
```

Creating an Encoded Password

DTP Engines can encrypt your password, which adds a layer of security to your interactions with DTP Server. Run the following command to print an encoded password:

```
-encodepass [MYPASSWORD]
```

Copy the encoded password that is returned and paste it into the jtestcli.properties file.

```
dtp.password=[ENCODED PASSWORD]
```

Connecting to Source Control

Parasoft DTP Engines ship with out-of-the-box support for the following SCMs:

Brand	Tested Version
AccuRev	4.6, 5.4, 6.2
ClearCase	2003.06, 7.0, 8.0
CVS	1.1.2
Git	1.7
Mercurial	1.8.0 - 3.6.3
Perforce	2006, 2012, 2013, 2014, 2015
Serena Dimensions	9.1, 10.1, 10.3 (2009 R2), 12.2
Star Team	2005, 2008, 2009
Subversion (SVN)	1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9

Brand	Tested Version
Synergy/CM	6.4, 7.0, 7.1
Microsoft Team Foundation Server	2008, 2010, 2012, 2013, 2015
Visual SourceSafe	5.0, 6.0, 2005

Edit the `jtestcli.properties` file located in the installation directory to connect to your SCM. Parameters will vary depending on the brand of your SCM. The following example shows the parameters required to connect to SVN:

```
scontrol.rep.type=svn
scontrol.rep.svn.url=https://svn_server/
scontrol.rep.svn.login=username
scontrol.rep.svn.password=password
scontrol.svn.exec=C:\\path\to\svn.exe
```

See “Customizing DTP Engines for Java”, page 61, for information about configuring your SCM connection.

Static Analysis Engine

Static Analysis Engine (SAE) enforces your coding policy with proven quality practices, such as static analysis and flow analysis, to ensure that your applications function as expected. The following sections describe how to analyze code with SAE.

- Basic Analysis
- Specifying Test Configurations
- Defining Test Scope
- Code Duplicate Analysis
- Using DTP Engines in an IDE

Basic Analysis

Run `jtestcli.exe` from the command line to launch the DTP Engine for Java. You must at minimum include a test configuration and specify the data file location to analyze code. See “Specifying Test Configurations”, page 14, and “Specifying Test Data Location”, page 11, for more information.

```
jtestcli.exe -settings settings.properties -config "builtin://Recommended Rules"
-data demo.data.json -report report
```

You can also launch the executable from dedicated plug-ins for supported build systems (Maven, Ant, and Gradle). See “Build Systems Integration”, page 87 for more information about build system plug-ins.

Add the $XTEST_ITX=714 environment variable to show arguments passed to `jtestcli.exe` executed by plug-ins.

Specifying Test Data Location

Use the `-data` switch to point directly to a `*.json` data file object generated by the build system plug-in. The automatically-generated `*.json` data file provides the same information as the workspace in a format specific to Static Analysis Engine.

```
-data file
```

Build system plug-ins automatically add the data argument and include the value they generate, so there is no need to add it to the plug-in configuration.

Settings Property Pattern

```
jtest.data=[path to file]
```

Data File Format

The data file (`jtest.data.json`) is a stream of json objects automatically generated and formatted by Static Analysis Engine build system plug-ins. It provides the same information about tested projects as the workspace. Each object in the file describes one project. Static Analysis Engine currently supports `classpath_project` common object type. See “Creating Project Files”, page 21, for an example.

Example Data File Object

```
{
          "type":       "classpath_project",
          "name":       "project_name",
          "location"    "/absolute/path/to/project",
          "compilations":     [
          {
               "sourcepath": [
          "/absolute/path/to/srcdir1"
               ],
               "classpath":     [
          "/absolute/path/to/classdir1",
          "/absolute/path/to/buildfile.jar"
               ],
               "bootpath":     [
          "absolute/path/to/java/ajavalib.jar"
          ],
               "encoding":     "project_encoding_name",
               "sourcelevel"   "project_source_level"
          },{
               ...
          }
      ]
}
{
     ...
}
```

Displaying Detailed Progress Information

Use the `-showdetails` switch to display detailed progress information.

```
-showdetails
```

Ant and Maven Pattern

```
<showdetails>true</showdetails>
```

Settings Property Pattern

```
console.verbosity.level=high
```

About Error Codes

Static Analysis Engine returns error exit codes (other than 0) when the following occurs:

- Static Analysis Engine is executed without a license.
- A test process exits with an internal exception.

- Command-line is malformed or refers to a resource that does not exist. See “Defining Test Scope”, page 18.

You can also use the `-fail` option to generate an exit code when the analysis reports static violations.

Specifying Test Configurations

Test configurations define how DTP Engines test and analyze code, including which static analysis rules are enabled, which tests to run, and other analysis parameters. DTP Engines ship with built-in test configurations, but users can create and store their own test configurations in the DTP server. You can access the DTP server via the DTP plug-in. If you have administrator-level access in DTP Report Center, you can also create test configurations directly in DTP (**administration> Engines> Test Configurations**).

User-defined test configurations can be downloaded from the DTP server and stored in the `[INSTALL_DIR]/configs/user` directory as `*.properties` files.

Use the `-config` switch to specify which test configuration to run:

```
-config "user://Configuration Name"
```

Ant and Maven Pattern

```
<config>user://Configuration Name</config>
```

Settings Property Pattern

```
jtest.config=user://Configuration Name
```

The test configuration being executed can be specified in the following ways (by default, the `builtin://Recommended Rules` test configuration is used):

Built-in Configurations

```
-config "builtin://Recommended Rules"
```

User-defined Configurations

```
-config "user://Foo Configuration"
```

DTP Server-hosted Configurations

```
-config "dtp://Foo Team Configuration"
-config "dtp://FooTeamConfig.properties"
```

Test configurations can also be referenced by filename and URL:

By File Name

```
-config "C:\Devel\Configs\FooConfig.properties"
```

By URL

```
-config "http://foo.bar.com/configs/FoodConfig.properties"
```

Viewing Available Test Configurations

Use the `-listconfigs` switch to print the available test configurations. Use arguments to filter configurations; the use of "*" expresssions is supported.

```
-listconfigs                    Prints all available configurations

-listconfigs builtin            Prints all built-in configurations

-listconfigs builtin*Secure*    Prints all built-in configurations that contain "secure" in
                                the name
```

```
-listconfigs
```

Built-in Test Configurations

The following table includes the test configurations shipped with DTP Engines in the `[INSTALL]/configs/builtin` directory.

Configuration Name	Description
Recommended Rules	The default configuration of recommended rules. Covers most Severity 1 and Severity 2 rules. Includes rules in the Flow Analysis Fast configuration.
Find Duplicated Code	Applies static code analysis rules that report duplicate code. Duplicate code may indicate poor application design and lead to maintainability issues.
Internationalize Code	Applies static code analysis to expose code that is likely to impede internationalization efforts.
Juliet 1.1 2011	
Metrics	Computes values for several code metrics.
New Features in JDK 1.5	
New Features in JDK 7	
DISA-STIG for Java	Includes rules that find issues identified in the DISA-STIG standard
Critical Rules	Includes most Severity 1 rules, as well as rules in the Flow Analysis Fast configuration.
Flow Analysis	Detects complex runtime errors without requiring test cases or application execution. Defects detected include using uninitialized or invalid memory, null pointer dereferencing, array and buffer overflows, division by zero, memory and resource leaks, and dead code. This requires a special Flow Analysis license option.

Configuration Name	Description
Flow Analysis Aggressive	Includes rules for deep flow analysis of code. Significant amount of time may be required to run this configuration.
Flow Analysis Fast	Includes rules for shallow depth of flow analysis, which limits the number of potentially acceptable defects from being reported.
Demo Configuration	Includes rules for demonstrating various techniques of code analysis. May not be suitable for large code bases.
Find Memory Problems	Includes rules for finding memory management issues in the code.
Find Unused Code	Includes rules for identifying unused/dead code.
CWE-SANS Top 25 [2009/2011]	Includes rules that find issues classified as Top 25 Most Dangerous Programming Errors of the CWE-SANS standard.
SAMATE NIST 2010	Includes rules that find issues identified in the NIST SAMATE standard
OWASP Top 10 [2007, 2010, 2013]	Includes rules that find issues identified in OWASP's Top 10 standard
PCI Data Security Standard	Includes rules that find issues identified in PCI Data Security Standard
Thread Safe Programming	Rules that uncover code which will be dangerous to run in multi-threaded environments— as well as help prevent common threading problems such as deadlocks, race conditions, missed notification, infinite loops, and data corruption.
Unit Tests	Includes the unit test execution data in the generated report file
Calculate Application Coverage	Processes the application coverage data to generate a coverage.xml file. See "Application Coverage", page 49, for additional information.
Unit Testing Best Practis ces	Helps you enforce unit testing best practices and ensure that assertions are made in your unit tests
Code Smells	Rules based on the Code Smells document (available at http://xp.c2.com/CodeSmell.html) by Kent Beck and Martin Fowler.

Configuration Name	Description
TDD	The TDD (Test Driven Development) configuration includes rules based on the Code Smells document (available at http://xp.c2.com/CodeSmell.html), rules that check whether the JUnit test classes are comprehensive for the tested class, and rules from the Critical Rules (Must Have) Test Configuration.

Creating Custom Rules

Use RuleWizard to create custom rules. To use the rule in the Static Analysis Engine, it needs to be enabled in a test configuration and the custom rule file must be located in one of the following directories:

- `[INSTALL_DIR]\rules\user\`
- `[DOCUMENTS DIR]\Parasoft\[engine]\rules` where `[DOCUMENTS DIR]` refers to the "My Documents" directory in Windows

Defining Test Scope

You can specify the input scope for your analysis, which includes the file or set of files to test, as well as the test data location. Use the `-resource` switch to specify the file or set of files for testing (also see “Specifying Test Data Location”, page 11, for information on pointing to the data location). You can use multiple `-resource` switches to specify multiple resources.

```
-resource [pattern]
```

Ant and Maven Pattern

```
<resource>pattern</resource>
```

Use the `<resources>` element to define multiple resources:

```
<resources>
    <resource>pattern1</resource>
    <resource>pattern2</resource>
</resources>
```

Settings Property Pattern

```
jtest.resource=pattern
```

Separate multiple resources with a comma:

```
jtest.resource=pattern1,pattern2
```

Resource Pattern Syntax

Resource patterns are matched to generated paths according to following convention: :

```
ProjectName/resource/inside/of/project/directory.extension
ProjectName/EXT/resource/inside/of/external/source/directories.extension
```

The `ProjectName` value is generated according to following rules:

- The Ant project name is taken from `name` property of `project` tag
- The Maven project name takes form of `groupId:artifactId`
- The Gradle project name is taken from `gradle.settings` file
- Project names configured in IDEs are fully maintained

You can customize the Maven and Gradle project names with the `projectNameTemplate` parameter. For details, see the build system plug-in manuals: [INSTALL]/manuals/ plugins-manual.html.

Generated paths include the `EXT` string if the specified scope includes external resources. For example, an analyzed project may include source files form another location that are linked to the project. In

the following example the `Simple.java` file is located within the Demo project, whereas the `Money.java` file has been added by linking and its locations is outside of the project:

```
Demo/src/examples/eval/Simple.java
Demo/EXT/examples/bank/Money.java
```

You can use Ant-style wildcards and other parameters to refine patterns. The following table describes supported usage:

Parameter	Description
`?`	Wildcard that matches one character (any character except path separaters)
`*`	Wildcard that matches zero or more characters (not including path separators)
`**`	Wildcard that matches zero or more path segments.
`/`	Separator for all operating systems
`"[non-alphanumeric characters]"`	Use quotation marks when resource paths contain spaces or other non-alphanumeric characters.

If the `-resource` argument only contains one value, the value is matched against the project name and a wildcard (`*`) will be used. The following table shows examples of `-resource` switch usage: .

Expression	Result
`-resource ProjectName/src`	Analyzes every file in the selected directory and subdirectories (without wildcards, the complete name of the folder has to be specified).
`-resource ProjectName/**/*.java`	Analyzes every Java file from selected project.
`-resource **/src/main/java/my/company/*.java`	Analyzes every Java file from the specific subdirectory of every project in current build.
`-resource ProjectName/src/main/java/my/company/File.java`	Analyzes the single specified resource.

Expression	Result

`-resource c:/resource.lst`	Analyzes the projects listed in the `resource.lst` file. Specify one project name per line:
	`ProjectName1`
	`ProjectName2`
	Provide the path to the file as a value to the `-resource` argument.

Fine-tuning the Scope

Use the `-include` and `-exclude` switches to apply additional filters to the scope.

- `-include` instructs Static Analysis Engine to test only the files that match the file system path; all other files are skipped.
- `-exclude` instructs Static Analysis Engine to test all files except for those that match the file system path.

If both switches are specified, then all files that match `-include`, but not those that match `-exclude` patterns are tested.

```
-include pattern
-exclude pattern
```

Ant and Maven Pattern

```
<include>pattern</include>
<exclude>pattern</exclude>
```

Settings Property Pattern

```
jtest.include=pattern
jtest.exclude=pattern
```

Use Ant-style wildcards and other parameters to with the `-include` and `-exclude` filters. The following table describes their usage:

Parameter	Description
`?`	Wildcard that matches one character (any character except path separaters)
`*`	Wildcard that matches zero or more characters (not including path separators)
`**`	Wildcard that matches zero or more path segments.
`/`	Separator
`path:`	Prefix for matching absolute disk path

You can specify a file system path to a list file (`*.lst`) to include or exclude files in bulk. Each item in the `*.lst` file is treated as a separate entry.

Examples

Usage	Description
`-include com/**`	Tests everything in packages that begin with "com".
`-include path:**/Bank.java`	Tests only Bank.java files
`-include path:C:/Project/src/**`	Tests all subfiles and subdirectories of `C:/Project/src`
`-include path:C:/Project/src/*`	Tests all files in `C:/Project/src`, but not subdirectories

`-include c:/include.lst`	Tests all files listed in `include.lst`. Each line is treated as a single pattern.
	Example:
	If `include.lst` contans the following lines:
	`**/*Account`
	`path:**/Bank.java`
	It has the same effect as the following command:
	`-include **/*Account`
	`-include path: **/Bank.java`

`-exclude **/internal/**`	Tests everything except classes that have "internal" as part of the package name.

Creating Project Files

You can use the `-project.*` switch to create a project file, which contains necessary information about executing DTP Engines. Use the `-project` switch when the solution is not provided. The switch can be used multiple times to analyze many projects. ANT-style wildcards are also supported, as well as paths to `*.lst` files. The following table describes the parameters for using `-project` switches.

Parameter	Description
`-project.location`	Enables generation of a file and indicates where the file will be placed after generation. All other `-project.*` options are ignored if the `-project.location` option is not set.
`-project.name`	Sets the project name.
`-project.encoding`	Sets the encoding of the project.
`-project.sourcepath`	Sets the source of project. Use this option multiple times to specify multiple source folders.
`-project.sourcelevel`	Specifies the Java compiler compliance level.
`-project.classpath`	Specifies the classpath.
`-project.javahome`	Used to find and add the library to the bootclasspath

Parameter	Description
`-project.class-path.jars.dir`	Specifies the path to the directory with jars that should be appended to the project classpath. View the json file to verify the order of the jars and ensure it meets your project requirements.
`-project.junit.outcomes`	Specifies the path to the XML file with JUnit test results that will be used to generate a report. Use this option multiple times to specify multiple report files.
`-project.compila-tion.classes`	Specifies the path to the compiled project classes that will be used to generate metadata information required to perform the "Calcu-late Application Coverage" configuration.

Example

```
-project.location C:/ExampleProject
-project.name ExampleProject
-project.encoding UTF-8
-project.sourcepath C:/ExampleProject/src
-project.sourcelevel C:/ExampleProject/src-test
-project.classpath C:/ExampleProject/lib/test.jar;C:/ExampleProject/lib/test2.jar
-project.javahome C:/Program Files/Java/jdk1.7
```

Configuring Authorship

You can configure DTP Engines to collect authorship data during analysis to facilitate task assignment. The data can be sent to the DTP server where additional analysis components, such as the Process Intelligence Engine (PIE), can be leveraged to facilitate defect remediation and development optimization.

You can configure DTP Engines to assign authorship based on information from source control, XML files that directly map sources to authors, and/or the current local user. You can also use the @author Javadoc tag.

About Authorship Configuration Priority

Authorship priority is determined by reading the settings in the .properties configuration file from top to bottom. If multiple authorship sources are used, the following order of precedence is used:

1. information from source control
2. XML map file
3. @author tags
4. current user

If one of the selected options does not determine an author (for instance, the @author tag was selected but the file does not have an @author tag), Authorship will be determined based on the next option selected. If an author cannot be determined, the user is set as "unknown". Likewise, if none of these options is selected, the user is set as "unknown."

Configuring How Authorship is Computed

Edit the `jtestcli.properties` configuration file to specify how authorship is determined:

```
scope.local=[true or false]
scope.scontrol=[true or false]
scope.xmlmap=[true or false]
scope.javadoc=[true or false]
```

Additional Authorship Configurations

By default, author names are case-sensitive, but you can disable case sensitivity:

```
authors.ignore.case=true
```

You can set the user name, email, and full name for a user with the `authors.user[identifyer]` setting. For example:

```
authors.user1=john,john.doe@company.com,John Doe
```

If a user is no longer on team or must transfer authorship to another user, you can use the `authors.mapping[x,y]` setting:

```
authors.mapping1=old_user,new_user
```

If you are transferring authorship between users, the author-to-author mapping information can be stored locally or in an a shared XML map file:

```
authors.mappings.location=[local or shared]
```

If the mapping file is shared, you must specify the location of the shared XML file:

```
authors.shared.path=[path to file]
```

Creating Authorship XML Map Files

The `<authorship>` element contains indicates the beginning of the mapping information.

The `<file />` element is placed inside the `<authorship>` element and takes two properties, `author` and `path` to map users to files or sets of files:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE authorship (View Source for full doctype...)>
    <authorship>
      <!-- assigns all files named: "foo/src/SomeClass.java" to "author1" -->
      <file author="author1" path="foo/src/SomeClass.java" />
```

You can use wildcards to map authors to sets of files. The following table contains examples:

Wildcard Expression	Description
`?oo/src/SomeClass.java`	Assigns all files that have names starting with any character (except /) and ends with "oo/src/"
`**.cs`	Assigns all *.cs files in any directory
`**/src/**`	Assigns every file whose path has a folder named "src"
`src/**`	Assigns all files located in directory "src"
`src/**/Test*`	Assigns all files in directory "src" whose name starts with "Test" (e.g., "src/some/other/dir/TestFile.c")

Mapping order matters. The mapping file is read from top to bottom, so beginning with the most specific mapping ensures that authorship will map to the correct files.

Suppressing Violations

Suppressions prevent DTP Engines from reporting additional occurrences of a specific static analysis task (multiple tasks might be reported for a single rule). Suppressions are useful when you want to follow a rule, but do not want to receive repeated messages about your intentional rule violations. If you do not want to receive error messages for any violations of a specific rule, disable the rule in the test configuration.

If you are using DTP Engines in an IDE, you can define suppressions using the GUI (see the DTP Plugin documentation for your IDE for details), otherwise suppressions are defined in the source code using the following syntax.

Line Suppression

```
<suppression keyword> [<rule category> | <rule category> . <rule id> | <rule
category > - <rule severity> | ALL ] <suppression comment>
```

Line Suppression Examples

```
// parasoft-suppress CODSTA "suppress all rules in category CODSTA"


// parasoft-suppress CODSTA.NEA "suppress rule CODSTA.NEA"


// parasoft-suppress CODSTA-1 "suppress all rules in category CODSTA with severity
level 1"


// parasoft-suppress ALL "suppress all rules"


// parasoft-suppress CODSTA FORMAT.MCH JAVADOC-3 "suppress all rules in category
CODSTA and rule FORMAT.MCH and all rules in category JAVADOC with severity level 3"
```

Block Suppression

```
<begin suppression keyword> [<rule category> | <rule category> . <rule id> | <rule
category > - <rule severity> | ALL ] <suppression comment>

 ..... source code block .....

<end suppression keyword> [<rule category> | <rule category> . <rule id> | <rule
category > - <rule severity> | ALL ] <suppression comment>
```

Block Suppression Examples

```
// parasoft-begin-suppress CODSTA "begin suppress all rules in category CODSTA"
.....
// parasoft-end-suppress CODSTA "end suppress all rules in category CODSTA"


// parasoft-begin-suppress CODSTA.NEA "begin suppress rule CODSTA.NEA"
.....
```

```
// parasoft-end-suppress CODSTA.NEA "end suppress rule CODSTA.NEA"


// parasoft-begin-suppress CODSTA-1 "begin suppress all rules in category CODSTA
with severity level 1"
......
// parasoft-end-suppress CODSTA-1 "end suppress all rules in category CODSTA with
severity level 1"


//parasoft-begin-suppress ALL "begin suppress all rules"
.....
// parasoft-end-suppress ALL "end suppress all rules"


// parasoft-begin-suppress CODSTA FORMAT.MCH "begin suppress all rules in category
CODSTA and rule FORMAT.MCH"
.....
// parasoft-end-suppress CODSTA FORMAT.MCH "end suppress all rules in category COD-
STA and rule FORMAT.MCH"


// parasoft-begin-suppress CODSTA "begin suppress all rules in category CODSTA"
.....
// parasoft-end-suppress CODSTA-1 "end suppress all rules in category CODSTA with
severity level 1; however rules with severity level 2-5 in category CODSTA are still
suppressed."
.....
// parasoft-end-suppress CODSTA "end suppress all rules in category CODSTA"


// parasoft-begin-suppress ALL "begin suppress all rules"
.....
// parasoft-end-suppress CODSTA FORMAT-1 "end suppress all rules in category CODSTA
and all rules in category FORMAT with severity level 1; however, others rules in COD-
STA and FORMAT-1 are still suppressed."
.....
// parasoft-end-suppress ALL "end suppress all rules"


//parasoft-begin-suppress ALL "begin suppress all rules, since no end suppression
comment, all rules will be suppressed starting from this line"
```

Code Duplicate Analysis

DTP Engines can check for duplicate code, which may indicate poor application design, as well as increase maintenance costs. During code duplication analysis, the code is parsed into smaller language elements (tokens). The tokens are analyzed according to a set of rules that specify what should be considered duplicate code. There are two types of rules for analyzing tokens:

- Simple rules for finding single token duplicates, e.g., string literals
- Complex rules for finding mulitple token duplicates, e.g., duplicate methods or statements

Run the Find Duplicated Code test configuration during analysis to execute code duplicates detection rules:

```
builtin://Find Duplicated Code
```

Flow Analysis

Flow Analysis is a type of static analysis technology that uses several analysis techniques, including simulation of application execution paths, to identify paths that could trigger runtime defects. Defects detected include use of uninitialized memory, null pointer dereferencing, division by zero, memory and resource leaks.

Since this analysis involves identifying and tracing complex paths, it exposes bugs that typically evade static code analysis and unit testing, and would be difficult to find through manual testing or inspection.

Flow Analysis' ability to expose bugs without executing code is especially valuable for users with legacy code bases and embedded code (where runtime detection of such errors is not effective or possible).

Run one of the Flow Analysis test configurations during analysis to execute flow analysis rules:

```
builtin://Flow Analysis Fast
builtin://Flow Analysis Standard
builtin://Flow Analysis Aggressive
```

Configuring Depth of Flow Analysis

Flow Analysis engine builds paths through the analyzed code to detect different kinds of problems. Since the analysis of all possible paths that span through the whole application may be infeasible, you can set up the desired level of depth of analysis. A deeper analysis will result in more findings, but the performance will be slower and the memory consumption will increase slightly.

You can specify the depth of analysis by using the test configuration interface in DTP. Go to **Report Center> Test Configurations> Static Analysis> Flow Analysis Advanced Settings> Performance> Depth of analysis** and choose one of the following options by selecting a radio button:

- **Shallowest (fastest)**: Finds only the most obvious problems in the source code. It is limited to cases where the cause of the problem is located close to the code where the problem occurs. The execution paths of violations found by this type of analysis normally span several lines of code in a single function. Only rarely will they span more than 3 function calls.
- **Shallow (fast):** Like the "Shallowest" analysis type, finds only the most obvious problems in the source code. However, it produces a greater overall number of findings and allows for examination of somewhat longer execution paths.
- **Standard**: Finds many complicated problems with execution paths containing tens of elements. Standard analysis goes beyond shallow analysis and also looks for more complicated problems, which can occur because of bad flow in a single function or due to improper interaction between different functions in different parts of the analyzed project. Violations found by this type of analysis often reveal non-trivial bugs in the analyzed source code and often span tens of lines of code.
- **Deep (slow)**: Allows for detection of a greater number of problems of the same complexity and nature as those defined for "Standard" depth. This type of analysis is slower than the standard one.
- **Thorough (slowest):** Finds more complicated problems. This type of analysis will perform a thorough scan of the code base; this requires more time, but will uncover many very complicated problems whose violation paths can span more than a hundred lines of code in different parts of the scanned application. This option is recommended for nightly runs.

The depth of Flow Analysis is set to **Standard** by default.

Setting Timeout Strategy

Apart from the depth of analysis, Flow Analysis engine uses an additional timeout guard to ensure the analysis completes within a reasonable time. An appropriate strategy can be set by using the test configuration interface in DTP. Go to **Report Center> Test Configurations> Static Analysis> Flow Analysis Advanced Settings> Performance> Strategy for Timeouts** and choose one of the following options by selecting a radio button:

- **time**: Analysis of the given hotspot is stopped after spending the defined amount of time on it. Note: in some cases, using this option can result in a slightly unstable number of violations being reported.
- **instructions**: Analysis of the given hotspot is stopped after executing the defined number of Flow Analysis engine instructions.Note: to determine the proper number of instructions to be set up for your environment, review information about timeouts in the Setup Problems section of the generated report.
- **off**: No timeout. Note: using this option may require significantly more time to finish the analysis.

The default timeout option is **time** set to 60 seconds. To get information about the Flow Analysis timeouts that occurred during the analysis, review the Setup Problems section of the report generated after the analysis.

Running Flow Analysis with Swapping of Analysis Data Enabled

In this mode, analysis data is written to disk. Swapping of analysis data uses the same persistent storage and is done in a similar process as incremental analysis. If analysis is run on a large project, the analysis data that represents a semantical model of the analyzed source code may consume all the memory available for running Flow Analysis. If this occurs, Flow Analysis will remove from memory parts of the analysis data that are not currently necessary and reread it from disk later.

In general, we recommend running Jtest in a large JVM heap configured with the Xmx JVM option. This is to minimize swapping, which results in greater performance. If sufficient memory is available, swapping of analysis data may be disabled, which may speed up code analysis. You can enable or disable the mode by using the test configuration interface in DTP:

Enable swapping of analysis data to disk: Enabled by default. If this option is disabled, it may result in faster analysis, if you are running Flow Analysis analysis on small to moderate size projects that do not require a lot of memory or when plenty of memory is available (for example, for 64-bit systems).

Configuring Verbosity of Flow Analysis

You can configure the following options by using the test configuration interface in DTP:

- **Do not report violations when cause cannot be shown:** Determines whether Flow Analysis reports violations where causes cannot be shown.

  Some Flow Analysis rules require that Flow Analysis checks all the possible paths leading to a certain point and verifies that a certain condition is met for all those paths. In such cases, a violation is associated with a set of paths (whereas in simple cases, a violation is represented by only one path). All of the paths in such a violation end with the violation point common to all the paths in the violation. However, different paths may start at different points in code. The beginning of each path is a violation cause (a point in code which stipulates a violation of a certain condition later in the code at the violation point). If a multipath violation's different paths have

different causes, Flow Analysis will show only the violation point (and not the violation causes).

Violations containing only the violation point may be difficult to understand (compared to regular cases where Flow Analysis shows complete paths starting from violation causes and leading to violation points). That's why we provide an option to hide violations where the cause cannot be shown.

- **Do not show more than one violation per point:** Restricts reporting to one violation (for a single rule) per violation point. For example, one violation will be reported when Flow Analysis detects a potential null dereference with multiple sources of the null value. When verbosity is set to this level, Flow Analysis performance is somewhat faster.

Null-checking Methods

The **Null-checking methods** option allows you to specify the expected return value when a null parameter is passed to a method. This reduces false positives and excessive paths that would normally be built when the return value for null variables are unknown.

Select the Enabled checkbox and provide the following information:

- **Fully-qualified type name (wildcard)**: the fully qualified name of the type that contains the method.
- **Method name (wildcard):** the name of the method.
- **Returned value when null:** the value that should be returned when a null parameter is passed to the method.
- **+ definitions in subclasses**: indicates whether the null-checking functions definitions in subclasses should be considered null-checking functions as well.

Null-checking Methods Example

Flow Analysis will analyze methods in the analysis scope. Null-checking method parameterizations are required when the methods are out of the scope of the current analysis (e.g. third-party libraries). The classes in the following example are defined in a different assembly and are outside the scope of the analysis (`UString`). The class uses static methods to manipulate code with Java strings in various ways, including the `UString.isEmptyOrNull(String)`, which is defined:

```
public class UString
{
    static boolean isEmptyOrNull(String variable)
    {
        if ((variable == null) || (variable.length() == 0)) {
            return true;
        }
        return false;
    }
}
```

And there is the following class, which is analyzed by Flow Analysis's BD-EXCEPT-NP rule:

```
public class SomeClass
{
    void someMethod(String variable)
    {
        boolean flag = variable == null; // violation search starts here
        if (UString.isEmptyOrNull(variable)) {
            /**
             * In case variable is null, we will always get to this branch of the if clause.
             */
            System.out.println("String is empty");
        } else {
```

```
            variable.toString(); // FALSE POSITIVE BD-EXCEPT-NP VIOLATION
        }
    }
}
```

Flow Analysis assumes that the return value for the method `isEmptyOrNull()` is unknown. This is because the method is not in the analysis scope. Flow Analysis will analyze the `then` and `else` branches of the `if` statement and find a violation in one of these statements. Calling this method, however, returns true when the variable is null.

By adding `UString.isEmptyOrNull()` to the list of methods in the Null-checking Methods parameterization with the specified return value, Flow Analysis will not report a violation if the tracked variable is passed to this method. This is because the wrong branch of the `if` statement will not be analyzed, resulting in an avoided false positive.

Null-checking Methods Restrictions

Methods added to this parameterization should be static methods that have a primitive boolean return value. This restriction is managed to avoid excessive complications in parameterization and ensuring that there are no other variables that could affect the result of the null-checking method.

Specifying Resources

The **Resources** tab allows you to define which resources the BD.RES category (Resources) rules should check. These rules check for the correct usage of all resources that are defined and enabled on this tab.

1. Specify the **Type of resource.**
2. Select the **Enabled** checkbox.
3. If appropriate/desired, disable the **Do not report leaks at termination** option.
4. Click the arrow to expand the **Resource Allocators** and **Resource Closers** tabs and complete the tables that open with the information about allocators and closers. Details about completing these tabs are provided below.

Configuring Resource Allocators

The **Resource allocators** table can be completed with the descriptors of methods that can produce a resource. The table has the following columns:

- **Enabled**: specifies whether the allocator should be considered during analysis.
- **Fully-qualified type name (wildcard)**: the fully-qualified name of the type where the method is declared. Use '*' if you want to describe a function declared in any type, or a global function declared outside of any type.
- **Method name (wildcard)**: the name of the allocating method. '*' can be used to denote any number of any symbols.
- **Resource parameters**: specifies that the method allocates a resource in one or more of its parameters. In this case, either specify a 1-based number of the parameter that is allocated by the method, or use '*' to denote that all of the parameters are allocated.
- **+ definitions in subclasses: a check box that indicates** whether the definitions (of methods with the given name) in subclasses should be considered allocators as well. Note that this applies to both instance and static methods.
- **"this" object is a resource**: a check box that indicates that the method allocates a resource in the object on which the method is called.

- **Returns a resource object**: a check box that inticates that a the method returns an allocated resource.
- **Return value constraint on error:** specifies a return value constraint in case of allocation failure if a resource allocator returns an integral value. Enter the condition in the following format: `<comparison operator><integer value>`. For example, if the function returns non-zero value on failure, enter "!=0" (without quotes) into the field. If return code on error is -1, type "==-1" there. In addition to "!=" and "==", you can use the following operators for specifying error conditions: ">", ">=", "<", and "<=".

It is common that allocation methods return an error code to indicate allocation failure. When an allocation method returns a resource, a NULL value normally indicates an allocation failure. When Flow Analysis is looking for resource leaks, it needs to understand if allocation succeeded or failed; this helps it report only missing calls to deallocation methods on paths where allocation actually occurred. In cases where a resource allocator method returns a resource, Flow Analysis assumes that the resource is successfully allocated if the returned value is not NULL.

Configuring Resource Closers

The **Resource closers** table can be completed with the descriptors of methods that can close a resource. The table has the following columns:

- **Enabled**: specifies whether the closer should be considered during analysis.
- **Fully-qualified type name (wildcard)**: the fully-qualified name of the type where the method is declared. Use '*' if you want to describe a function declared in any type , or a global function declared outside of any type.
- **Method name (wildcard)**: the name of the closing method. '*' can be used to denote any number of any symbols.
- **+ definitions in subclasses: a check box that indicates** whether the definitions (of methods with the given name) in subclasses should be considered closers as well. Note that this applies to both instance and static methods.
- **"this" object is a resource**: a check box that indicates that a resource in the object on which the method is called is closed.
- **Resource parameters**: specifies that a resource in one or more of its parameters is closed. In this case, either specify a 1-based number of the parameter that is closed by the method, or use '*' to denote that all of the parameters are allocated.

Metrics Analysis

DTP Engines can compute several code metrics, such as code complexity, coupling between objects, and lack of cohesion, which can help you understand potential weak points in the code. Run the Metrics test configuration during analysis to execute metrics analysis rules:

```
builtin://Metrics
```

Metrics analysis is added to the HTML and XML report files generated by DTP Engines. See “Metrics Summary”, page 39, for information about reports.

Setting Metrics Thresholds

You can set upper and lower boundaries so that a static analysis violation is reported if a metric is calculated outside the specified value range. For example, if you want to restrict the number of logical lines in a project, you could configure the Metrics test configuration so that a violation is reported if the Number of Logical Lines metric exceeds the limit.

The Metrics test configuration shipped with DTP Engines includes default threshold values. There are some rules, such as Number of Files (METRIC.NOF), for which thresholds cannot be set.

Metric thresholds can be set using the following methods:

- By using the test configuration interface in DTP (see "Report Center> Test Configurations> Editing Test Configurations> Metrics Tab" in the Development Testing Platform user manual for details).
- By editing the test configuration using the interface in an IDE (see "Working with Test Configurations> Creating Custom Test Configurations" in the DTP Plugin manual for your IDE).
- By manually editing the test configuration file:
    1. Duplicate the built-in Metrics test configuration ([INSTALL]/configs/builtin) to the user configurations directory ([INSTALL]/configs/user)
    2. Open the duplicate configuration in an editor and set the `[METRIC.ID].ThresholdEnabled` property to `true`.
    3. Configure the lower and upper boundaries in the `[METRIC.ID].Threshold` property according to the following format:

        ```
        [METRIC.ID].Threshold=l [lower boundary value] g [upper boundary value]
        ```
    4. Save the test configuration and run the analysis using the custom metrics test configuration.

Using DTP Engines in an IDE

You can use DTP Engines within IntelliJ IDEA, NetBeans or Eclipse . Integrating with an IDE gives you a desktop interface for executing code analysis locally, viewing results, and leveraging the data and test configurations stored in DTP server. You can also import findings from DTP Server into your development environment.

This integration is achieved with the DTP Plugin for IntelliJ IDEA, NetBeans or Eclipse and the DTP Engine Plugin. See **Parasoft> Online Help** in your IDE menu bar (recommended) or *Jtest Desktop User Guide* shipped with your product for installation, usage, and other details.

Reporting

DTP Engines print results to the output console, as well as save an HTML report to the `[WORKING_DIR]/reports` directory by default. Data for the HTML report is stored in the directory as an XML file, which can be used for importing results into a supported Parasoft DTP Plugin for the IDE and Parasoft DTP Plugin for Java (see "Parasoft DTP Plugin for [IDE] User's Guide" for additional information). For an overview of the HTML report structure, see "Viewing Reports", page 35.

If the engines are connected to DTP, reports are also sent to the server (see "Sending Results to Development Testing Platform (DTP) Server", page 41).

File paths that are displayed in reports follow the resource pattern syntax, see "Resource Pattern Syntax", page 18.

Specifying Report Output Location

You can use the `-report` switch during analysis to specify an output directory for reports.

```
-report location
```

Ant and Maven Pattern

```
<report>location</report>
```

Settings Property Pattern

```
report.location=location
```

You can also use the `report.location` property to change the location of an HTML report.

```
report.location=<HTML_REPORT_LOCATION>
```

Specifying Report Format

You can also generate a PDF report or a report using a custom extension to the specified directory by setting the `report.format` property. See "Report Settings", page 70, for additional information.

```
report.format=pdf
```

Viewing Reports

Open the report.html or report.pdf file saved to the working directory or location specified with the `-report` switch. Reports may contain different sections depending on the type of analysis, but the following sections are included in all static and flow analysis configurations.

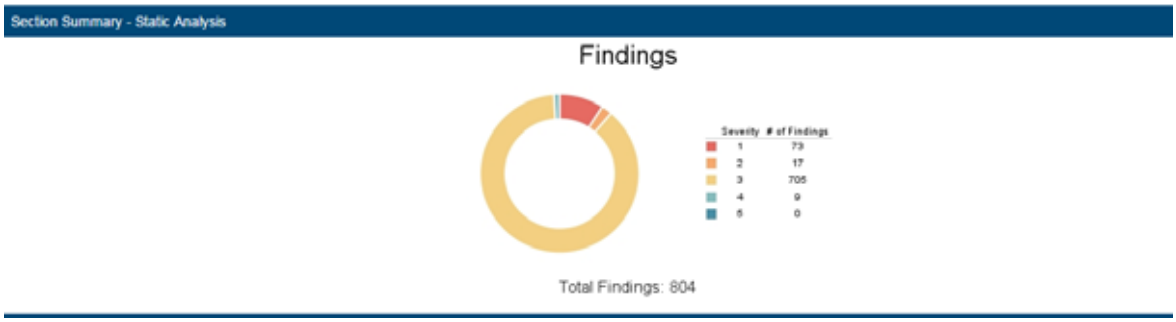
Header

The following information is included:

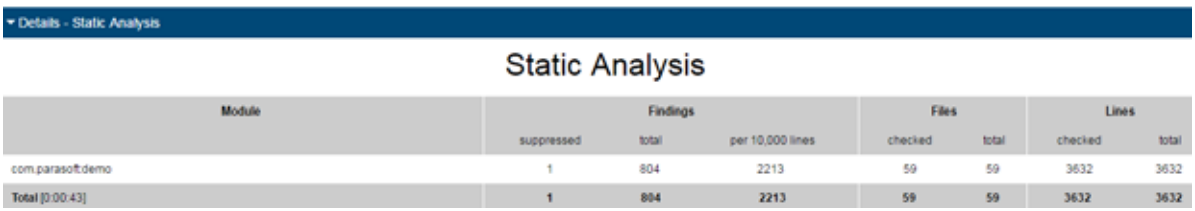
- Tool used for the analysis
- Build ID
- Test configuration
- Time stamp of the analysis
- Machine name and user name
- Session tag
- Project name
- Number of findings with the highest severity
- Number of failed tests

Static Analysis

The first part of the report covers the Static Analysis findings and is divided into two main sections.The first section is a summary which shows an overview of findings displayed as a pie chart. The colors indicate different severity types and their corresponding number of findings detected during static analysis.



The second section shows the details of static analysis findings. It starts with a table which includes static analysis results.



The following information is included:

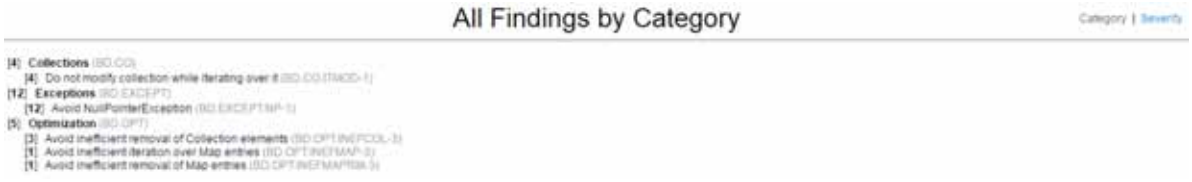
- Name of module

- Number of suppressed rules
- Total number of findings
- Average number of findings per 10,000 lines
- Number of analyzed files
- Total number of files in the module
- Number of code lines analyzed
- Total number of code lines in the module

All Findings

The All Findings section displays the details of findings organized by category or severity. Click the **Severity** or **Category** link to toggle between views.

In category view, findings are reported by rule and grouped by category. A count of how many times each rule was violated in the scope of analysis is also shown.



In severity view, findings are reported and grouped by severity. A count of findings per severity is also included.

These sections are merged in PDF versions of the report.

Findings by Author

This section includes a table of authors associated with the analyzed code and a count of findings per each author. Findings are segmented into findings associated with suppressed rules and findings recommended for remediation. Click on an author link to view their finding details.

The details view includes the following information:

- File containing the finding and its location
- Violation message and rule
- Flow analysis reports also mark the cause of the violation (C), violation points (P), thrown exceptions (E), and important data flows (!)

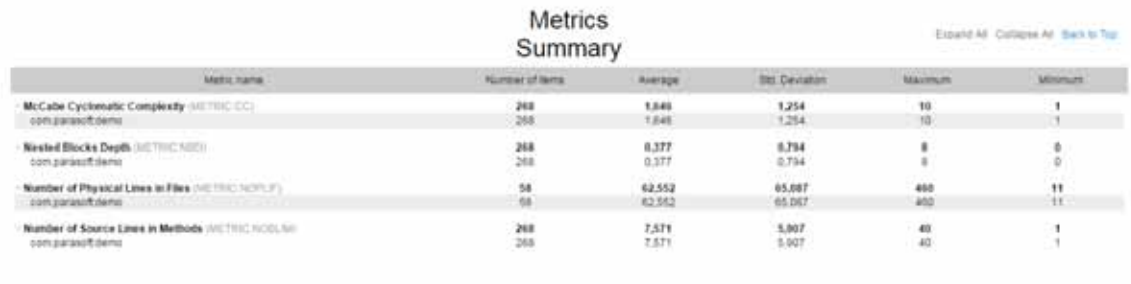
Findings by File

You can navigate the analyzed code to the reported findings in the Findings by File section. Each node begins with a value that indicates the total number of findings in the node. The value in brackets shows

the number of suppressed rules in the node. You can click nodes marked with a plus sign (+) to expand them. PDF versions of the reports are already fully expanded.

Metrics Summary

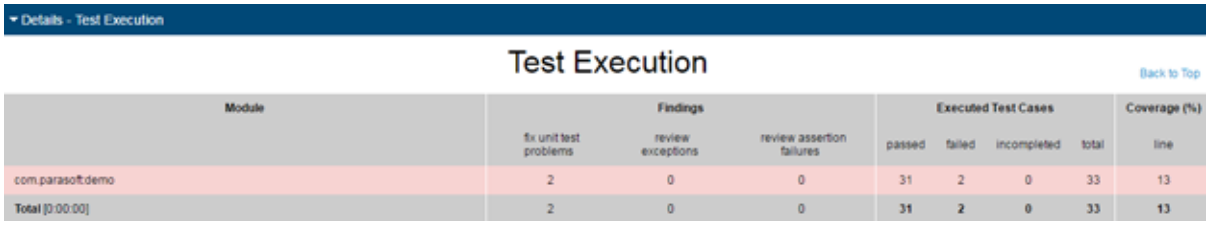
If your test configuration includes metrics analysis, a metrics section will appear in the report. See “Metrics Analysis”, page 33, for additional information.



Test Execution

The second part of the report covers the Test Execution results and is divided into two sections.The first section is a summary which shows an overview of test failures and coverage displayed as pie charts.

The second section shows the details of test execution. It starts with a table which includes test execution results and coverage information.


Details - Test Execution

Test Execution

Back to Top

Module	Findings			Executed Test Cases				Coverage (%)
	fix unit test problems	review exceptions	review assertion failures	passed	failed	incompleted	total	line
com.parasoft.demo	2	0	0	31	2	0	33	13
Total [0:00:00]	2	0	0	**31**	**2**	**0**	**33**	**13**

The following information is included:

- Module name
- Number of unit test problems which need to be fixed
- Number of exceptions which need to be reviewed
- Number of assertion failures which need to be reviewed
- Number of unit tests successfully executed
- Number of unit tests failures
- Number of incomplete unit tests
- Total number of unit tests
- Line coverage expressed as percentage

All Findings

The All Findings section displays the details of all unit test problems detected during test execution.


All Findings

[2] **Unit Test Problems**
[2] **Assertion Failures**
[1] junit.framework.AssertionFailedError
[1] java.lang.AssertionError

Findings by Author

This section includes a table of authors associated with the analyzed code and shows the total number of findings for each author. Click on an author link to view their finding details.


Findings by Author

Back to Top

Author	Findings	
	total	recommended
annstu	2	0

annstu Total Findings : 2

Back to Top

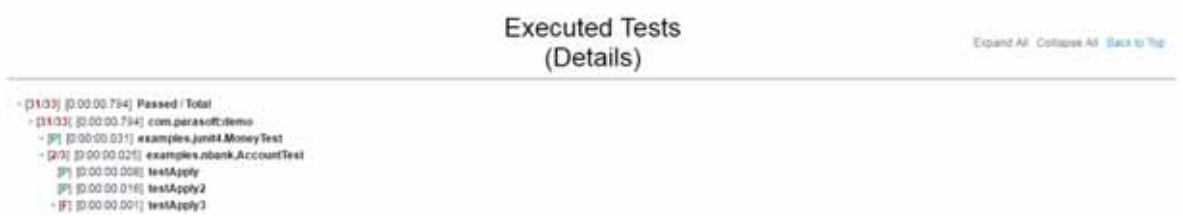
/demo/tests/examples/stubs/InterpreterTest.java [1]
Test case: testAdd34
value is:-1 (failure)
org.junit.Assert.fail(Assert.java:88)
org.junit.Assert.assertTrue(Assert.java:41)
examples.stubs.InterpreterTest.testAdd34(InterpreterTest.java:43)

The details view includes the following information:

- Finding location
- Test name
- Failure message

Executed Tests (Details)

You can view the findings in the Executed Tests (Details) section. The nodes where all the test passed are marked with "P" in square brackets. The nodes with test failures begin with a set of values in square brackets. The first value is a count of successfully passed tests and the second indicates the total number of tests executed in the node.The letter "F" indicates the final node where the test failed. You can click nodes marked with a plus sign (+) to expand them.



Coverage

This section shows the details of coverage collected during the test execution. Each node starts with a set of values. The first value shows coverage expressed as percentage. The second value is a count of the number of lines in the node which were covered during the test execution. The third value indicates the total number of lines in the node. You can click nodes marked with a plus sign (+) to expand them.

Test Parameters

The arguments specified during analysis are shown in the Test Parameters section.

Sending Results to Development Testing Platform (DTP) Server

See “Connecting to DTP Server”, page 8, for information about configuring your connection to DTP Server. Use the `-publish` switch to report test results to DTP server.

```
-publish
```

Ant and Maven Pattern

```
<publish>true</publish>
```

Settings Property Pattern

```
report.dtp.publish=true
```

Publishing Source Code to DTP Server

By default, tested sources are sent to DTP when the report setting is enabled. This enables DTP to present source code associated with findings.

You can use the `report.dtp.publish.src` setting to disable the publishing of source code, restrict the depth of source code publishing, or enable source code publishing when sending reports to DTP Server is disabled. See “Settings Reference”, page 63, for additional information on DTP Engine settings.

The `report.dtp.publish.src` setting takes one of the following values:

- `off`: Code is not published to DTP server.
- `min`: Publishes minimal part of sources. Only source code that has no reference to source control is published.
- `full`: Publishes all sources associated with the specified scope. This is the default settings.

See the "Development Testing Platform User Guide" for additional information about viewing source code in DTP.

Publishing Sources to DTP Without Running Code Analysis

DTP Engines need to execute to send data to DTP Server, but you may want to send sources without running analysis.

1. Create an empty test configuration and save it to `[INSTALL_DIR]/configs/user` (see “Specifying Test Configurations”, page 14).
2. Run the configuration with appropriate `report.dtp.publish.src` setting.

Unit Test Connector

Unit Test Connector (UTC) allows you to run unit tests created in open source unit testing tools and report results to DTP. UTC for Java currently ships with out-of-the-box support for the following unit testing tools:

- JUnit 3 and 4
- TestNG

Visit the Parasoft Marketplace (http://marketplace.parasoft.com) for additional unit test tool integrations.

Framework Support Details

The following table describes detailed support for testing frameworks:

Framework	Supported	Unsupported
JUnit 3	Test classes directly extending `junit.framework.TestCase` with `test*` methods.	Suite classes: classes directly extending `junit.framework.TestCase` and providing `static suite()` methods, regardless of test name changes. Test classes directly extending `junit.framework.TestCase` with the `runTest()` method.
JUnit 4	Test methods annotated with `@org.junit.Test` annotation.	Tests parameterized with `@org.junit.runners.Parameterized.Parameters` Theories with `@org.junit.experimental.theories.Theory`
TestNG	Test methods annotated with `@org.testng.annotations.Test` annotation. XML reports (JUnit format only) will be processed.	Tests parameterized with `@org.testng.annotations.DataProvider` Classes annotated with `@org.testng.annotations.Test annotation` Tests parameterized with `@org.testng.annotations.Factory` XML reports in TestNG format won't be processed.

Running JUnit Tests

JUnits are run by Maven, Ant or Gradle frameworks. Results are collected and reported using dedicated plug-ins for each framework. For details on how to use the plug-ins, including how to troubleshoot issues with Maven, Ant, or Gradle, see manuals/plugins-manual/index.html.

Tagging Unique Test Runs

Use the `session.tag` property to define a tag that can be assigned to results from a specific test run. The tag is a unique identifier for the analysis process on a specific module. Using the same session tag overwrites data with results from the most recent run. By default, the tag is set to the name of the executed test configuration.

```
session.tag=[name]
```

Associating Tests with Development Artifacts

You can configure DTP Engines to associate tests with a broad range of development artifacts, such as requirements, defects, tasks, and feature requests.

To successfully associate unit tests with artifacts, you need to:

1. Enable the artifact association property.
2. Specify issue tracking tags and configure their URL associations.
3. Use the tags in the Javadoc.

See the sections below for details.

Enabling Artifact Associations

Set the `report.associations` property to `true` to enable associations with artifacts. This also enables/disables test associations in the HTML report.

```
report.associations=true
```

Specifying Issue Tracking Tags

The following tags for artifact types are associated by default when report associations is enabled:

- pr (defects)
- fr (enhancements)
- task
- asset
- req (user stories)

You can use the `issue.tracking.tags` property to define any number of additional tracking tags. Separate the tags' names with a comma:

```
issue.tracking.tags=tag1,tag2,tag3
```

Configuring Issue Tracking Tags and URL Associations

You can generate a link to the association in the HTML report:

```
report.assoc.url.tag1=[URL]
```

URLs can contain `[%ID%]` or `${id}` variables, which will be replaced by issue identifiers. For example:

```
report.assoc.url.tag1=http://bugzilla.company.com/show_bug.cgi?id=[%ID%]
```

Enabling Test Details

You can enable or disable showing test details in the HTML report:

```
report.contexts_details=[true | false]
```

The `report.contexts_details` property must be set to `true` to enable showing associations.

The product's property file is preconfigured to enable showing test details.

See “Report Settings”, page 70 for additional information.

Using Javadoc Tags

Use Javadoc tags to associate unit tests with artifacts.

Place the tag in the Javadoc to associate it with your tests. The tags used in the Javadoc should be preceded by a @ character.

```
/**
 * @bug 12345
 * @pr 223344
 * @tag1 5533
 */
@Test
public void testSomething()
{
   ...
}
```

You can also associate a tag with a class. As a result, it will be associated with all the tests within this class. In the example below, tag 9876 is associated with both tests within the Test class, whereas tag 111 is associated only with the testSomething2 test.

```
/**
 * @tag 9876
 */
public class Test {

    testSomething1()
    {
     ...
    }

    /**
     * @tag 111
     */
    testSomething2()
    {
      ...
    }
}
```

Multiple Associations

You can associate one tag with more than one artifact. Separate multiple associations with a comma that is not followed by a space character.

```
/**
 * @task 1234,2345
 */
@Test
public void testSomething()
{
   ...
}
```

If you separate the tasks with a comma and a white space, the test will be associated only with the first listed artifact. In the example below, the test is associated only with task 1234:

```
/**
 * @task 1234, 2345
 */
@Test
public void testSomething()
{
   ...
}
```

Collecting Coverage

Coverage from unit tests is collected during JUnit test execution. See “Code Coverage Engine”, page 47 for information on collecting code coverage and reporting it to DTP.

Code Coverage Engine

In this section:

Coverage for Unit Tests

You can collect coverage information during unit test execution by running special commands on the following build systems:

- Maven
- Ant
- Gradle

Associating coverage information collected during unit test execution is supported for JUnit 4 only.

For details on how to configure your build system plug-in, as well as how to execute tests and collect coverage during testing, see the build systems plug-in manual: [INSTALL]/manuals/plugins-manual/index.html.

In order to collect coverage information and send it to DTP, run the built-in Unit Tests configuration during test execution. Maven is used in the following example:

```
mvn clean test-compile jtest:instrument jtest:jtest -Djtest.config="builtin://Unit Tests"
```

See “Unit Test Connector”, page 43, for information on setting up and executing unit tests.

Merging Coverage Data

In order to properly merge coverage data in DTP, you must specify one or more coverage image tags in the command line or .properties settings file. The coverage image(s) is automatically sent to the connected DTP server where it can be associated with a filter.

You can specify a set of up to three tags that can be used to create coverage images in DTP Server with the `report.coverage.images` property:

```
report.coverage.images=[tag1; tag2; tag3]
```

Associate coverage images in DTP in the Report Center administration page (**administration> Projects> Filters> [click on a filter]**).

You can also use the `report.coverage.limit` property to specify a lower coverage threshold:

```
report.coverage.limit=[value]
```

Coverage results lower than this value are highlighted in the report. The default value is `40`.

Application Coverage

You can monitor and collect coverage data during manual or automated functional tests performed on a running web application server. You can also send coverage data and test results to DTP, which merges and correlates the data. The application coverage information can be displayed in the DTP Coverage Explorer (see the "Coverage Explorer" chapter in the DTP user manual), which provides insights about how well the application is tested, as well as the quality of your tests.

Prerequisites

The following components are required for collecting coverage:

- Java JDK 1.5
- Apache Maven, Gradle or Ant build system

Java JDK 8 line break calculations in bytecode are inconsistent with previous Java versions. This may lead to inconsistencies in coverage results.

Process Overview

The DTP Engine for Java ships with a component called the coverage agent. The coverage agent is attached to the application under test (AUT) and monitors the code being executed as the AUT runs. When the coverage agent is attached to the AUT, a REST API is exposed that enables you to mark the beginning and end of each test and test session.

Metadata about the lines of code that can be covered (static coverage data) is collected by running a dedicated test configuration as part of the application build process. During test execution, interactions with the coverage agent are written to a dynamic coverage map, which contains markers that specify which lines of code were touched.

The DTP Engine processes the dynamic coverage map and static coverage data. A coverage.xml file, which contains the coverage information, is produced and sent to DTP. When DTP receives the coverage data, it's loaded into a coverage image, which is a special tag that enables you to aggregate coverage data from runs with the same build ID. The coverage image enables you to associate coverage information with specific tests.

Test results are also sent to DTP from the tool executing the tests (i.e., SOAtest, tests executed by the DTP Engine, manual tests, etc.) in a report.xml file. If the build IDs for the coverage data file and the report match, DTP is able to correlate the data and display the coverage information.

The complete process is detailed in the following sections.

Configuring the Application Under Test for Coverage

There are a few processes for preparing the AUT:

1. The static coverage file must be generated. The static coverage file contains metadata about user classes, methods, and lines. This is described in “Generating the Static Coverage File”, page 50.
2. The coverage agent must be attached to the AUT. See “Attaching the Coverage Agent to the AUT”, page 50.
3. The coverage agent includes default settings for outputting files, determining scope, etc., but you can set properties in a configuration file to meet your application coverage goals. See “Configuring the Coverage Agent”, page 51.

Generating the Static Coverage File

The package that contains the static coverage file is created during the build process by the Jtest Maven, Gradle or Ant plugin. It must be generated on the build machine that contains the source code. The static coverage file can be used until the code changes.

If you use a source control system, ensure that your source control settings are properly configured; see “Settings Reference”, page 63.

Execute the following command in the AUT’s main directory to generate the package that contains the static coverage file with Maven:

```
mvn package jtest:monitor
```

or Gradle:

```
gradle assemble jtest-monitor -I [INSTALL]/integration/gradle/init.gradle
```

Ant requires all classes to be compiled before the monitor task is executed. Modify your project prior to the build and configure the task to ensure the correct sequence. The following example shows how the target can be configured:

```
<target name="jtest-monitor" depends="compile">
            <jtest:monitor/>
</target>
```

Execute the following command in the AUT’s main directory to build the project and generate the package that contains the static coverage file:

```
ant -lib [INSTALL]/integration/ant/jtest-ant-plugin.jar -listener
  com.parasoft.Listener jtest-monitor
```

The monitor.zip package will be generated and placed into the build output directory. The path to the location will be printed on the console.

The package contains the following:

- static_coverage.xml - this file contains static coverage information
- agent.jar - Jtest Java coverage agent jar archive
- agent.properties - agent settings file that contain scope parameters generated during the build process and other attributes
- agent.sh/agent.bat - script that generates the Jtest Java agent VM arguments necessary for attaching the agent to the AUT process

Attaching the Coverage Agent to the AUT

Extract the contents of the monitor.zip package to the server machine and run the agent.sh/agent.bat script to generate the Jtest Java agent VM arguments.

The scripts from the monitor package output the `-javaagent` VM argument to the console. You will need this argument to connect the agent to the AUT, which will result in the runtime_coverage subdirectory:

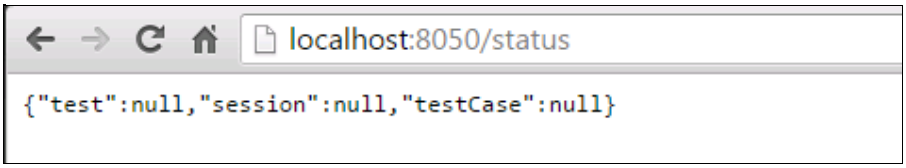
```
Jtest Agent VM argument:
-javaagent:"[path to agent dir]\agent.jar"=settings="[path to agent properties
file]\agent.properties",runtimeData="[path to monitor dir]\monitor\
runtime_coverage"
```

Add the `-javaagent` flag to the application server's startup script and restart the server. See "Step 5a: Collecting Runtime Data from Apache Tomcat", page 57, "Step 5b: Collecting Runtime Data from JBoss and WildFly", page 58, and "Step 5c: Collecting Runtime Data from Oracle WebLogic Server", page 60 in the Web Application Coverage Tutorial below.

Attaching the coverage agent to the AUT exposes a REST API for controlling the agent. Enter the following URL into the browser bar to verify that the coverage agent is set:

[application_host]:8050/status/

If the application has been properly configured, the API will return data:



Configuring the Coverage Agent

Application servers usually contain more than one application. Additionally, common server classes or application libraries do not need to be instrumented. The Jtest DTP Engine only needs to collect coverage for application source code. Instrumenting all classes would be too time-consuming.

The application on the server is already built, so we cannot gather information about which classes come from the source code. For this reason, properly setting the scope of the coverage agent is very important.

You can configure the coverage agent with the agent.properties file or arguments to `-javaagent`. The agent.properties file is generated in the monitor.zip package (see "Generating the Static Coverage File", page 50). It contains properties that can be modified to properly configure the coverage agent. The following example shows how the coverage agent can be configured with the agent.properties.file:

```
jtest.agent.runtimeData=[path to runtime_coverage directory]
jtest.agent.includes=com/myapp/data,com/myapp/common/**
jtest.agent.excludes=com/myapp/transport/*,com/myapp/autogen/**
jtest.agent.autostart=false
```

Adding properties to `-javaagent` requires modifying their names by removing the "jtest.agent" prefix, for example:

```
-javaagent:"[path to agent dir]\agent.jar"=settings="[path to agent properties
file]\agent.properties",autostart=true
```

Properties provided as arguments to `-javaagent` override properties configured in the agent.properties file. In the above examples, the `autostart=true` will override `jtest.agent.autostart=false`.

The following table describes all properties that can be used to configure the coverage agent:

Property	Description
`jtest.agent.runtimeData`	Where the runtime data will be stored. The following example will create files in the in 'C:/tmp/myapp/' directory with `runtime_`' as the name prefix: `'C:/tmp/myapp/runtime'`
`jtest.agent.includes`	Coma separated list of patterns that specify classes to be instrumented. The following wildcards are supported: `*` matches zero or more characters `**` matches multiple directory levels In the following example, all classes from the `com.myapp.data` package and all classes from package and subpackages that start with `com.myapp.common` will be instrumented: `com/myapp/data/*,com/myapp/common/**`
`jtest.agent.excludes`	Coma separated list of patterns that specify classes to be *excluded* from instrumentation. The following wildcards are supported: `*` matches zero or more characters `**` matches multiple directory levels In the following example, all classes from the `com.myapp.transport` package and all classes from package and subpackages that start with `com.myapp.autogen` will be excluded from instrumentation: `com/myapp/transport/*,com/myapp/autogen/**`
`jtest.agent.autostart`	Enables/disables automatic runtime data collection; default is `true`
`jtest.agent.port`	Sets up agent communication port; default is `8050`
`jtest.agent.debug`	Enables/disables verbose output to console; default is `false`
`jtest.agent.collect-TestCoverage`	Enables/disables collecting coverage information for test cases. The default value is `false`.
`jtest.agent.enableMul-tiuserCoverage`	Enables/disables collecting web application coverage for multiple users; the default value is `false`. Setting this property to `true` allows you to collect multi-user coverage with Coverage Agent Manager. See the "Coverage Agent Manager (CAM) section of the DTP documentation for details.

Test Configuration and Execution

You can use SOAtest to run functional tests (refer the Application Coverage chapter of the SOAtest documentation to set up the test configuration), as well as execute manual tests. At the end of the test session, coverage will be saved in `runtime_coverage_[timestamp].data` files in the directory speci-

fied in SOAtest. This information will eventually be merged with the static coverage data to create a coverage.xml file and uploaded to DTP.

Uploading Test Results to DTP

For tests executed by SOAtest, the SOAtest XML report will need to be uploaded to DTP. See the "Uploading Rest Results to DTP" section in the Application Coverage topic in the SOAtest documentation for details.

Generating a Dynamic Coverage Data File and Uploading it to DTP

The following settings should be configured in the jtestcli.properties file in order to properly merge coverage data.

- `report.coverage.images` - this setting specifies a set of tags that are used to create coverage images in DTP Server. A coverage image is a unique identifier for aggregating coverage data from runs with the same build ID. DTP supports up to three coverage images per report.
- `session.tag` - this specifies a unique identifier for the test run and is used to distinguish different runs on the same build.
- `build.id` - this setting specifies a build identifier used to label results. It may be unique for each build, but it may also label several test sessions executed during a specified build.

These settings are in addition to the other properties that must be configured, such as scope, authorship, and DTP settings. See the following sections:

- “Connecting to DTP Server”, page 8
- “Sending Results to Development Testing Platform (DTP) Server”, page 41
- “Settings Reference”, page 63

In order to fill the coverage.xml file with runtime coverage data, the DTP Engine must have access to the runtime coverage data generated during test execution, as well as the static coverage data. Run the Calculate Application Coverage test configuration using the following arguments to provide the necessary data:

```
jtestcli -staticcoverage [path to static_coverage.xml file] -runtimecoverage
[path/dir] -config "builtin://Calculate Application Coverage"
```

Known Limitations

- If multiple users are simultaneously accessing the same web application, the coverage data they collect may be mixed. To ensure that coverage is properly associated with individual users, the mutliuser mode must be enabled (see “Configuring the Coverage Agent”, page 51).
- The HTTP or HTTPS protocols are required to enable the multiuser mode, as the user-specific information must be provided with the HTTP header.
- In the multiuser mode, the "default" user (the user who has not specified their ID) may collect extra coverage information from other users who are accessing the same web application.
- In the multiuser mode, assigning coverage collected for multithreaded application to individual users is limited. Coverage data for child threads is not assigned to the user who is actually accessing the application, but to the "default" user.

- Coverage data collected for web application initialization is not assigned to a specific user, but to the "default" user.

Reviewing Coverage in DTP

You can use the Coverage Explorer in DTP to review the application coverage achieved during test execution. See the DTP documentation for details on viewing coverage information.

Web Application Coverage Tutorial

The following tutorial will guide you through collecting application coverage with the Jtest DTP Engine. It includes information about basic topics, such as running the server and deploying applications, so that beginners and advanced users can benefit.

Prerequisites

The following requirements are only necessary to complete this tutorial. Web Application Coverage can be collected on any server that can accept Java Agents.

- Java JDK 1.5
- Apache Maven build system
- Calculator example, which is extremely basic web application. You can find this example in the [INSTALL]/examples directory
- One of the following application servers:
    - Apache Tomcat (this scenario has been verified with versions 6.0, 7.0, 8.0)
    - JBoss/WildFly
    - Oracle WebLogic

Step 1: Preparing Calculator Example

The application must be packaged into a WAR file (Web Application Archive) on the server. Use the maven-war-plugin shipped with the engines to package the application. The plugin automatically builds applications with the correct WAR structure. Execute the following command in the application directory:

```
mvn clean install
```

When the project finished building, the WAR file will be placed into the target directory: Calculator\target\Calculator.war

Step 2a: Deploying Example to Tomcat Application Servers

Before proceeding with this step, you must build the WAR file as described in “Step 1: Preparing Calculator Example”, page 54. If you are using a JBoss/WildFly application server, skip to “Step 2b: Deploying Example to WildFly/JBoss Application Servers”, page 55. If you are using a WebLogic application server, skip to “Step 2c: Deploying Example to Oracle WebLogic Server”, page 56

There are many ways to deploy application servers, but in this tutorial we demonstrate how to deploy remotely using the Tomcat Manager web application.

1. Add a new user to the server by opening [Apache Tomcat Installation Directory]/conf/tomcat-users.xml and add the following line in the `<tomcat-users>` section:

```
<user username="tomcat" password="tomcat" roles="tomcat, manager-gui"/>
```

2. Run the server by executing the following command:

   Windows

```
[Your apache-tomcat installation directory]/bin/startup.bat
```

   Linux

```
[Your apache-tomcat installation directory]/bin/startup.sh
```

3. Open the following URL in a browser and enter the username and password set in step 1 (tomcat/tomcat):

```
http://localhost:8080/manager/html
```

4. In the WAR file to deploy section, click **Choose File** and browse to Calculator.war
5. Click **Deploy**; the application will be available at the following URL:

```
http://localhost:8080/calculator
```

6. Interact with the application to as a sanity test.

Step 2b: Deploying Example to WildFly/JBoss Application Servers

There are many ways to deploy WildFly/JBoss application servers, but we demonstrate deployment in this tutorial by using the JBoss web console. These instructions can be applied to JBoss AS 7.1.1, JBoss EAP 6.4, and WildFly 8.2 and 9.0 servers.

1. Add a new user to the server by running the add-user script on Windows:

```
[JBoss installation directory]/bin/add-user.bat
```

2. Choose Management User when prompted to choose a type of user:

```
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a
```

3. Enter a username and password when prompted leaving the Realm field blank:

```
Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : admin
Password : 1adm-adm-adm
```

4. Run the server by executing the following script:

```
[JBoss installation directory]/bin/standalone.bat
```

5. Open the following URL in a browser and enter the username and password specified in step 3 (admin/1adm-adm-adm):

```
http://localhost:9990/console/App.html#deployments
```

6. Click **Add** (or **Add Content** for JBoss 7.1.1) and browse to the Calculator.war file.
7. Click **Next** and click **Save**.
8. Enabled the Calculator.war file in the Deployments list and open the following URL in a browser:

```
http://localhost:8080/Calculator
```

9. Interact with the application to as a sanity test.

Step 2c: Deploying Example to Oracle WebLogic Server

There are many ways to deploy WebLogic application server, but in this tutorial we demonstrate how to deploy remotely using the Admin Console. This scenario has been verified with versions 12.2.1 and 10.3.6.

This tutorial assumes that you have already installed the WebLogic server and created your server domain.

1. Go to **Administration Console** (http://localhost:7001/console).
2. In the **Domain Structure** panel menu, click **Deployments** to open the **Summary of Deployments** section.
3. In the **Deployments** table, click the **Install** button.
4. Specify the path to your WAR file in th **Arguments** field.
5. Click **Next**.
6. Choose the **Install this deployment as an application** option.
7. Click **Next**.
8. Click **Finish**.
9. Interact with the application to test its functionalities (http://localhost:7001/Calculator)

Step 3: Preparing Metadata Files

Execute the following command in the example application's main directory:

```
mvn package jtest:monitor
```

The goal generates the monitor.zip package, which contains artifacts necessary for collecting application coverage. See "Generating the Static Coverage File", page 50, for details.

Step 4: Generating the `javaagent` VM Argument

Extract the contents of the monitor.zip package to the server machine and run the agent.bat (Windows) or agent.sh (Linux) script. This will generate the `javaagent` flag that will be printed to the console.

Your `javaagent` flag may resemble the following:

```
-javaagent:"E:\Parasoft\JTest\examples\calculator\target\jtest\monitor\monitor\
agent.jar"=settings="E:\Parasoft\JTest\examples\calculator\target\jtest\monitor\
monitor\agent.properties",runtimeData="E:\Parasoft\JTest\examples\calculator\
target\jtest\monitor\monitor\runtime_coverage"
```

Step 5a: Collecting Runtime Data from Apache Tomcat

If you are using a JBoss/WildFly server, skip to "Step 5b: Collecting Runtime Data from JBoss and WildFly", page 58. If you are using a WebLogic server, skip to "Step 5c: Collecting Runtime Data from Oracle WebLogic Server", page 60s

1. Open the script file:

   Windows

```
[Your apache-tomcat installation directory]/bin/catalina.bat
```

   Linux

```
[Your apache-tomcat installation directory]/bin/catalina.sh
```

2. Place the `javaagent` flag at the beginning of the script:

   Windows

```
if "%1"=="stop" goto skip_instrumentation
set JAVA_OPTS=%JAVA_OPTS% [generated javaagent flag]
:skip_instrumentation
```

   Linux

```
if [ "$1" = "start" -o "$1" = "run" ]; then
export JAVA_OPTS="$JAVA_OPTS [generated javaagent flag]"
fi
```

   The `-javaagent` flag must be placed in a single line.

3. Restart the server and open the following URL in a browser: .

```
http://localhost:8080/Calculator
```

4. Interact with the application and stop the server

The Jtest Agent will write runtime data according to the `runtimeData` property of the `-javaagent` flag generated by the `mvn package jtest:monitor` goal. By default, the runtime data be written to the `[path to monitor dir]/monitor/runtime_coverage` directory.

Step 5b: Collecting Runtime Data from JBoss and WildFly

Standard Java hierarchical class loaders are not suitable for J2EE servers because all JAR files are always loaded, whether they are used or not; the application cannot load JAR files on demand. Additionally, an adequate solution for restricting visibility between JAR files is not available, which frequently causes conflicts between two versions of one library. This also leads to the creation of big class loaders that contain everything.

JBoss and WildFly servers use JBoss Modules to overcome these challenges. The implementation of modular, non-hierarchical class loading is also the basis of JBoss OSGi and JBoss Java EE implementations.

The advantages of this class loader, however, introduces new problems in terms of collecting runtime data from these servers. The Jtest Agent library must be visible to application classes, but it is prevented by JBoss Modules. This is one reason for adding the `-javaagent` flag. Failing to do so will return `ClassNotFoundException`.

For this reason, the `jboss.modules.system.pkgs` system property should also be used. But there is another problem: JBoss sets this flag in its own scripts. This behavior complicates integration between JBoss and the Jtest Agent because these scripts may overwrite previously set scripts.

JBOss AS 7.1/JBoss EAP 6.4/WildFly 8.2/9.0

1. Add the `-javaagent` flag to the startup script:

   Windows:

   a. Open `[Your jboss installation directory]/bin/standalone.conf.bat` and add the `javaagent` flag to JAVA_OPTS at the end of the file:

```
set "JAVA_OPTS=%JAVA_OPTS% [generated javaagent flag]"
```

   Leave `":JAVA_OPTS_SET"` as the last line in the file.

   Place the `-javaagent` flag on one line.

   Add the `-XX:-UseSplitVerifier` flag to JAVA_OPTS for servers working on Java 7.

   Add the `-noverify` flag to JAVA_OPTS for servers on working on Java 8.

   b. Add `javaagent` classes to `jboss.modules.system.pkgs` settings by changing the following section from:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman"
```

   to

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byte-
man,com.parasoft.jtest.instrumentation,shaded.com.parasoft.jtest.runtime"
```

c. Restart the server using the `standalone.bat` script.

Linux:

a. Open `[Your jboss installation directory]/bin/standalone.conf` and add the `javaagent` flag to JAVA_OPTS at the end of the file:

```
set "JAVA_OPTS=$JAVA_OPTS [generated javaagent flag]"
```

Place the `-javaagent` flag on one line.

Add the `-XX:-UseSplitVerifier` flag to JAVA_OPTS for servers working on Java 7.

Add the `-noverify` flag to JAVA_OPTS for servers on working on Java 8.

b. Add `javaagent` classes to `jboss.modules.system.pkgs` settings by changing the following section from:

```
JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman"
```

to

```
JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman,com.parasoft.jtest.instrumenta-
tion,shaded.com.parasoft.jtest.runtime"
```

c. Restart the server using the `standalone.sh` script.

2. Open to the following URL in a web browser:

```
http://localhost:8080/Calculator/
```

3. Interact with the application and stop the server.

The Jtest Agent will create a `runtimeData.data` file in the Jtest target maven directory (target/jtest).The location is written into the .json data file, so you do not need to remember it.

JBOss 6.1/5.1

The `jboss.modules.system.pkgs` property does not need to be set for older versions of JBoss.

Windows:

1. Open `[Your jboss installation directory]/bin/run.conf.bat` and add the `javaagent` flag to JAVA_OPTS at the end of the file:

```
set "JAVA_OPTS=%JAVA_OPTS% [generated javaagent flag]"
```

Leave `":JAVA_OPTS_SET"` as the last line in the file.

Place the `-javaagent` flag on one line.

2. Restart the server using the `run.bat` script.

Linux:

1. Open `[Your jboss installation directory]/bin/run.conf.bat` and add the `javaagent` flag to JAVA_OPTS at the end of the file:

```
set "JAVA_OPTS=$JAVA_OPTS [generated javaagent flag]"
```

Place the `-javaagent` flag on one line.

2. Restart the server using the `run.sh` script.

Step 5c: Collecting Runtime Data from Oracle WebLogic Server

1. Open the script file:

   Windows

```
[Your weblogic installation directory]/user_projects/domains/[your domain]bin/
startWebLogic.cmd
```

   Linux

```
[Your weblogic installation directory]/user_projects/domains/[your domain]bin/
startWebLogic.sh
```

2. Add the `-javaagent` flag to JAVA_OPTIONS (in the section that starts with the START WEBLOGIC comment). The `-javaagent` flag must be placed in a single line:

   Windows

```
@REM START WEBLOGIC
set JAVA_OPTIONS=%JAVA_OPTIONS% [generated javaagent flag]
```

   Linux

```
# START WEBLOGIC
export JAVA_OPTIONS=$JAVA_OPTIONS [generated javaagent flag]
```

3. Restart the server and open the following URL in the browser: http://localhost:7001/Calculator
4. Interact with the application.
5. Stop the server

The Jtest Agent will write runtime data according to the `runtimeData` property of the `-javaagent` flag generated by the `mvn package jtest:monitor` goal. By default, the runtime data be written to the `[path to monitor dir]/monitor/runtime_coverage` directory.

Step 6: Produce Coverage Report

Run the Calculate Application Coverage test configuration and pass the static_coverage.xml file and directory to the runtime coverage data using the dedicated `-staticcoverage` and `-runtimecoverage` parameters:

```
jtestcli -staticcoverage [path to static_coverage.xml file] -runtimecoverage
[path/dir] -config "builtin://Calculate Application Coverage"
```

The path to the static_coverage.xml file must point to the extracted contents of the monitor.zip package, which contains this file.

The path to the runtime_coverage directory is printed to the console as the `runtimeData` property of the `-javaagent` VM argument.

Customizing DTP Engines for Java

Default settings are specified in the `jtestcli.properties` file located in the [INSTALL_DIR] directory. You can use the `-settings` switch followed by an absolut path to a settings file to customize Static Analysis Engine. You can use `-settings` multiple times to specify several properties files.

```
jtestcli -settings path/to/settings.properties
-settings path/to/another/settings.properties
```

Settings configured in the `jtestcli.properties` file are read first and any references specified in additional files will be overwritten. You can also store common settings in a `jtestcli.properties` file in the $USER_HOME directory to overwrite settings stored in the properties file in the [INSTALL_DIR] directory.

General settings are applied in the following order:

1. `[INSTALL_DIR]/etc/jtestcli.properties`; the base configuration file for Static Analysis Engine and **should not be modified**.
2. `[INSTALL_DIR]/jtestcli.properties`; contains templates for commonly used settings (license, reporting etc.)
3. `[USER_HOME]/jtestcli.properties`; optional
4. `[WORKING_DIR]/jtestcli.properties;` optional
5. Custom settings passed with the command line switch `-settings path/to/settings.properties` (e.g., `-settings ../settings.properties`)
6. Custom settings passed with the command line switch `-property [key=value]`

All of the above settings can be overridden by custom settings that are passed with command line switches (e.g. `-report`, `-config`, `-dtp.share.enabled`).

Ant and Maven Pattern

```
<settings>path/to/settings.properties</settings>
```

Settings Hierarchy for Maven

If you use Maven, you can configure `jtestcli` settings with:

- `-Djtest.[Maven property name]` (e.g., `-Djtest.settings="my.general.properties"`).
- the `pom.xml` file
- `-Dproperty.jtest.[property name]` (e.g., `-Dproperty.jtest.license.use_network=true`)

Both `-Djtest` and `pom.xml` override settings that are passed with `-Dproperty.jtest`.

`-Djtest` overrides settings in `pom.xml` if they are provided as user properties. However, if a setting value is specified directly in `pom.xml`, it has the highest priority and cannot be overridden by command line settings. For example, the test configuration specified in pom.xml as a user property `<config>${jtest.config}</config>` can be overridden with `-Djtest.config`. If it is hardcoded as `<config>builtin://Demo Configuration</config>`, cannot be overridden.

See the Jtest Goal page in plugins-manual.html for the complete list of parameters.

You should keep all user-level customizations, including custom settings, license, rules, test configurations, compiler configurations, outside of the DTP Engine installation directory so they are not affected by reinstallations or updates.

Viewing Current Settings

Use the `-showsettings` option to print the current settings and customizations, including the origin file for each configuration.

Ant and Maven Pattern

```
<showsettings>true</showsettings>
```

Settings Property Pattern

```
jtest.showsettings=true
```

Using Variables

The following table shows variables that can be used in settings values.

> We recommend you avoid using spaces, +, /, or any other special characters when setting variables or values for configuration settings, as some API calls may require properly encoded URLs.

Variable	Description	Example
`analysis_type`	Outputs a comma separated list of enabled analysis types (e.g., Static, Generation and Execution)	`${analysis_type}`
`env_var`	Outputs the value of the environmental variable specified after the colon.	`${env_var:HOME}`
`config_name`	Outputs the name of executed Test Configuration.	`${config_name}`
`dtp_project`	Outputs the name of DTP project specified in the settings file using `dtp.project` option.	`${dtp_project}`
`project_module`	Outputs the name of the tested project's module. If more than one module is provided as an input, the first tested module name is output followed by an ellipsis (...). The variable can be configured in the settings file with the `project.module` option.	`${module_name}`
`host_name`	Outputs the name of the host.	`${host_name}`
`user_name`	Outputs the name of the current user.	`${user_name}`
`os`	Outputs the name of the operating system.	`${os}`
`arch`	Outputs the name of the operating system architecture	`${arch}`
`exec_env`	Outputs the execution environment. Ths variable is a concatenation of `${os}` and `${arch}` variables. It can be configured in the sittings file with the `exec.env` option.	`${exec_env}`

Variable	Description	Example
`scontrol_branch`	Outputs the source control branch name for the tested project. If more than one branch name is detected, the first branch name is output followed by an ellipsis (...). The variable can be configured in the settings file with the `scontrol.branch` option.	`${scontrol_branch}`
`tool_name`	Outputs the name of the tool (i.e., Jtest, C++test, dotTEST).	`${tool_name}`
`jvm_prop`	Outputs the value of the Java vm property specified after the colon.	`${jvm_prop:os.name}`

Settings Reference

The following tables contain settings that are currently supported in DTP Engines.

Base Configuration Settings

Setting	Value	Description/Notes

`console.verbosity.level`	`low`
	`normal`
	`high`
		Specifies the verbosity level for the Console
		`low`: configures the Console view to show errors and basic information about the current steps and status (done, failed, up-to-date).
		`normal`: (default) also shows command lines and issues reported during test and analysis.
		`high`: also shows warnings.

`parallel.mode`	`disabled`
	`auto`
	`manual`
		Determines which of the following modes is active:
		`disabled`: configures Parasoft Test to use only one of the available CPUs.
		`auto`: (default) allows Parasoft Test to control parallel processing settings.
		`manual`: allows you to manually configure parallel processing settings to suit your specific needs.

`parallel.no_memory_limit`	`true` `false`	Enables/disables restrictions (beyond existing system limitations) on the memory consumed by parallel processing. Default is `false`

Setting	Value	Description/Notes
parallel.free_memory_limit	[percentage]	Specifies the amount of memory that should be kept free in low memory conditions (expressed as a percentage of the total memory available for the application). This is used to ensure that free memory is available for other processes. Default is 25
parallel.max_threads	[number]	Specifies the maximum number of parallel threads that can be executed simultaneously. The actual number of parallel threads is determined by the number of CPUs, available memory, and license settings. The default value is equal to the number of CPUs
file.encoding.mode	default auto user	Specifies how file encoding is determined. default: enables use of system properties auto: enables automatic detection of encoding for the Far-East languages specified with file.encoding.lang user: enables use of specified encoding by file.encoding.name.
file.encoding.lang	[code]	Allows specify language's numeric code when file.encoding.mode is set to auto: Japanese = 1 Chinese = 2 Simplified Chinese = 3 Traditional Chinese = 4 Korean = 5
file.encoding.name	[encoding]	Allows you to specify the encoding name when file.encoding.mode is set to user: ASCII-US UTF-8 UTF-16 UTF-16LE UTF-16BE …
settings.validation	true false	Enables/disables settings validation.

Setting	Value	Description/Notes
settings.rules.file.jtest	path	Indicates the path to a file that contains additional rules for settings validation. The file should follow the .properties format and include rules according to the following examples: engine.path=$ANY engine.enabled=$BOOLEAN engine.analysis.deep=$INTEGER engine.severity.limit=$REGEXP{[1-5]} engine.verbosity.level=$REGEXP_IC{(low)|(normal)|(high)}

Test Configuration Settings

Setting	Value	Description/Notes
configuration.dir.builtin	[path]	Path to directory with built-in test configurations.
configuration.dir.user	[path]	Path to directory with user-defined test configurations.
configuration.share.path	[path]	Path on DTP server share with shared test configuration.
jtest.custom.rule.dir	[path to directory]	Specifies the location of user-defined coding standard rules. Default is [INSTALL_DIR]/rules/user

Development Testing Platform Settings

Setting	Value	Description/Notes
dtp.server	[host]	Specifies the host name of the DTP server.
dtp.port	[port]	Specifies the port number on DTP server port. The default settings is 443.
dtp.user dtp.password	[username] [password]	Specifies authentication to connect to DTP server.
dtp.project	[project_name]	Specifies the name of the DTP project that you want linked to. This settings is optional.
dtp.autoconfig	true false	Enables auto configuration with settings stored on the DTP server. The default is false.

Setting	Value	Description/Notes
report.dtp.publish	true false	Determines whether the current installation is reporting test results to DTP server. The default is false.
report.dtp.publish.src	off min full	Determines whether tested source code is published to DTP server. off: code is not published to DTP server. min: publishes minimal part of sources. In most cases, source code without references to source control, e.g., auto-generated code, is published. full: publishes all sources associated with the specified scope. The default is full if report.dtp.publish is enabled, otherwise the default is off
dtp.share.enabled	true false	Enables/disables connection to Team Server. The default is false.
jtest.license.use_network	true false	Enables/disables license retrieval from License Service. The default setting is true.
jtest.license.network.type	dtp ls	Sets the network license type. dtp:file count license that limits usage to a certain number of files as determined by your licensing agreement ls: floating license (machine locked) that limits usage to a certain number of machines
jtest.license.local.password	[password]	Specifies the local license password.
jtest.license.local.expiration	[expiration]	Specifies the local license expiration date.
jtest.license.network.edition	desktop_edition server_edition custom_edition	Specifies the type of license that will be retrieve from License Service for this installation. Default is custom_edition
jtest.license.custom_edition_features	[feature_name, ...]	Specifies active features for custom license edition.

Setting	Value	Description/Notes
jtest.license.wait.for.tokens.time	[minutes]	Specifies the time that tool will wait for a license if a license is not currently available.

Scope and Authorship Settings

Setting	Value	Description/Notes
scope.local	true false	Enables/disables code authorship computation based on the local user and system files modification time. Default is true
scope.scontrol	true false	Enables/disables code authorship computation based on data from a supported source control system. Default is false
scope.javadoc	true false	Enables/disables code authorship computation based on javadoc @author tag. Default is true
scope.xmlmap	true false	Enables/disables task assignment based on an XML mapping file that defines how tasks should be assigned for particular files or sets of files. See “Creating Authorship XML Map Files”, page 24 for syntax information. Default is false
scope.xmlmap.file	[path]	Specifies the path to XML mapping file that defines how tasks should be assigned for particular files or sets of files.
authors.ignore.case	true false	Enables/disables author name case sensitivity. Example: true: David and david are considered the same user. Default is false

Setting	Value	Description/Notes
authors.mappings.location	local shared	Specifies where the authorship mapping file is stored. Default is local. See authors.user and authors.mapping options for details. When set to shared, mappings could be specified in file located on DTP share. See authors.shared.path option for details.
authors.shared.path	[path]	Specifies the location of authors mapping file in DTP share. Example: authors.shared.path=xtest/authors_map.txt
authors.user{n}	[user_name, email, full_name]	Specifies a specific author by user name, email, and full name. Example: authors.user1=dan,dan@parasoft.com,Dan Stowe authors.user2=jim,jim@parasoft.com,Jim White
authors.mapping{n}	[from_user, to_user]	Specifies a specific author mapping. Example: authors.mapping1=old_user,new_user authors.mapping2=broken_user,correct_user

Suppression Settings

Setting	Value	Description/Notes
suppression{n}.file.ext	[ext]	Specifies the extension of types of files that should be scanned for comment suppressions. Example: suppression1.file.ext=xml suppression2.file.ext=java Set the comment prefix with the suppression{n}.comment setting.

Setting	Value	Description/Notes

`suppression{n}.comment`	`[comment]`	Specifies comment prefix for types of files identified in `suppression.file.ext` setting.
		Example:
		`suppression1.comment=//`
		`suppression2.comment=<!--`

`suppression{n}.comment.suffix`	`[comment suffix]`	Defines the suppression comment suffix when file extensions has been specified with the `suppression.file.ext` setting. If not specified then suppression comments will not be suffixed. Example: `suppression1.comment.suffix=-->`
`suppression{n}.block.only`	`true|fales`	Enables/disables block-only comment suppressions support when file extensions have been specified with the `suppression.file.ext` setting. Default is `false`.

`suppression.local.dir`	`[path]`	Specifies the custom directory for storing local suppressions. An absolute path should be provided.
		Example:
		`suppression.local.dir=file:///C:/parasoft/suppression/storage`
		`suppression.local.dir=../suppression/storage`

Technical Support Settings

Setting	Value	Description/Notes
`techsupport.enabled`	`true` `false`	Enables/disables global automatic technical support data collection is globally enabled with verbose logging. Default is `false`

Setting	Value	Description/Notes
`logging.verbose`	`true` `false`	Enables/disables verbose logs. Verbose logs are stored in the `xtest.log` file in the location specified with the `local.storage.dir` setting. Verbose logging state persists across sessions (restored on application startup). The log is a rolling file with a fixed maximum size. A backup is created whenever the max size is reached . Default is `false`
`logging.scontrol.verbose`	`true` `false`	Enables/disables output from source control commands in verbose logs. Note that output from source control may include fragments of analyzed source code. Default is `false`
`techsupport.create.on.exit`	`true` `false`	Enables/disables automatic archive creation when the application is shut down. The `techsupport.enabled` setting must also be enabled for packages to be created automatically. Default is `true`.
`techsupport.archive.location`	`[path]`	Specifies the custom directory where support packages should be created.
`techsupport.include.reports`	`true` `false`	Enables/disables the inclusion of reports in the technical support package.

Report Settings

Setting	Value	Description/Notes
`session.tag`	`[name]`	Specifies a tag for signing results form the test session. The tag is a unique identifier for the specified analysis process made on a specified module. Reports for different test sessions should be marked with different session tags.

Setting	Value	Description/Notes
`build.id`	`[id]`	Specifies a build identifier used to label results. It may be unique for each build but may also label more than one test sessions that were executed during a specified build. The default settings is `build-yyyy-MM-dd HH:mm:ss`
`project.module`	`[name]`	Specifies a custom name for the project's module. The setting may be used to describe unique runs. If unspecified, the tested module is detected automatically based on code provided to analysis.
`exec.env`	`[env1;env2...]`	Specifies a list of tags that describe the environment where the run was executed. Tags may describe operating system (e.g., Windows, Linux), architecture (e.g., x86, x86_64), compiler, browser, etc. The `exec.env` tags enable the entire session to be described. A detailed description of the environment may also be included in the test suite, test, or test case levels via services API.
`report.location`	`[path]`	Specifies the directory where report should be created.

`report.format`	`xml`
	`html`
	`pdf`
	`csv`
	`custom`
		Specifies the report format. Use a comma separated list of formats to generate multiple formats.
		Default is `xml`

`report.custom.extension`	`[ext]`	Specifies the report file extension of the XSL file for a custom format. Use with `report.format=custom` and `report.custom.xsl.file`.
`report.custom.xsl.file`	`[path]`	Specifies the location of the XSL file for a custom format. Use with `report.format=custom` and `report.custom.extension`

Setting	Value	Description/Notes
`report.developer_errors`	`true` `false`	Determines whether manager reports include details about developer errors. The default is `true`.
`report.developer_reports`	`true` `false`	Determines whether the system generates detailed reports for all developers (in addition to a summary report for managers). The default is `false`.
`report.authors_details`	`true` `false`	Determines whether the report includes an overview of the number and type of tasks assigned to each developer. The default is `true`.
`report.contexts_details`	`true` `false`	Determines whether the report includes an overview of the files that were checked or executed during testing. The default is `true`.
`report.suppressed_msgs`	`true` `false`	Determines whether report includes suppressed messages. The default setting is `false`.
`report.metadata`	`true` `false`	Determines whether additional metadata about findings should be downloaded from DTP. Only findings that are already present on DTP are affected. The DTP server must also support the metadata service for this settting to have an effect. Default is `true`.

`report.scontrol`	`off`
	`min`
	`full`
		Specifies if and how much additional information from source control is included in the report:
		`min`: repositories, file paths and revisions
		`full`: includes the same information as min, as well as task revisions and comments.
		Default is `off`

`report.associations`	`true` `false`	Enables/disables showing requirements, defects, tasks, and feature requests associated with a test in the report. The default is `true`.

Setting	Value	Description/Notes
issue.tracking.tags	[tag1,tag2,...]	Specifies a list of issue tracking tags. The following tags are supported by default: pr, fr, task, asset, req.
report.assoc.url.[tag]	[url]	Generates link to association inside the HTML report. The URL is a query string containing an [%ID%] placeholder for the PropertyAttribute value.
report.active_rules	true false	Determines if report contains a list of the rules that were enabled for the test. The default setting is true.
report.rules	[url]	Specifies a directory for storing static analysis rules HTML files (retrieved by clicking the **Printable Docs** button in the Test Configuration's Static Analysis tab). Examples: report.rules=file:///C:/parasoft/gendoc/ report.rules=../gendoc/
report.test_params	true false	Determines whether report includes test parameter details. The default setting is true.
report.coverage.images	[tag1,. . .]	Specifies a set of tags that will be used to create coverage images in DTP Server. DTP supports up to 3 coverage images per report.
report.coverage.limit	[limit]	Value that specifies the lower coverage threshold. Coverage results lower than this value are highlighted in the report. Default is 40
report.metrics.attributes	[attr1;attr2;...]	Specifies a list of additional attributes for metric results. The following attributes are supported by default: module, namespace, type, method.
report.archive	true false	Enables/disables archiving reports into a ZIP file.

Setting	Value	Description/Notes
report.graph.start_date	[MM/dd/yy]	Specifies start date for trend graphs that track static analysis task, test execution, and coverage. Use with report.graph.period=[?d|?m|?y]
report.graph.period	[?d|?m|?y]	Determines the duration from the start date for trend graphs that track static analysis task, test execution, and coverage. Use with report.graph.start_date=[MM/dd/yy]
report.mail.enabled	true false	Enables/disables report emails to developers and additional recipients specified with the report.mail.cc setting. If enabled, all developers that worked on project code will automatically be sent a report that contains the errors/results related to his or her work. The default setting is false.
report.mail.server	[host]	Specifies the mail server used to send reports.
report.mail.port	[port]	Specifies the port for SMTP server. The default port is 25.
report.mail.security	[security]	Specifies SMTP server connection security. STARTTLS and SSL are supported. The default is STARTTLS.
report.mail.subject	[subject line]	Specifies the subject line of the emails sent.
report.mail.username report.mail.password report.mail.realm	[user_name] [password] [realm]	Specifies the settings for SMTP server authentication. The realm value is required only for those servers that authenticate using SASL realm.
report.mail.domain	[domain]	Specifies the mail domain used to send reports.
report.mail.time_delay	[time]	Specifies a time delay between emailing reports (to avoid bulk email restrictions).
report.mail.from	[email|user_name]	Specifies the "from" line of the emails sent.

Setting	Value	Description/Notes
`report.mail.attachments`	`true` `false`	Enables/disables sending reports as attachments. All components are included as attachments; before you can view a report with images, all attachments must be saved to the disk. The default setting is `false`.
`report.mail.compact`	`trends` `links`	Specifies how report information is delivered in the email. `trends`: email contains a trend graph, summary tables, and other compact data; detailed data is not included. `links`: email only contains a link to a report available on DTP server. This setting is not configured by default
`report.mail.format`	`html` `ascii`	Specifies content type for the email. The default setting is `html`.
`report.mail.cc`	`[email; ...]`	Specifies email address for sending comprehensive manager reports. Multiple addresses must separated with a semicolon. This setting is commonly used to send reports to managers or architects, as well as select developers.
`report.mail.include`	`[email, ...]`	Specifies email addresses of developers that you want to receive developer reports. Multiple addresses must separated with a semicolon. This setting is commonly used to send developer reports to developers if developer reports are not sent automatically (e.g., because the team is not using a supported source control system). This setting overrides addresses specified in the 'exclude' list.
`report.mail.exclude`	`[email; ...]`	Specifies email addresses that should be excluded from automatically receiving reports.

Setting	Value	Description/Notes
report.mail.exclude.developers	true false	Enables/disables report emails to developers not explicitly listed in the report.mail.cc setting. This setting is used to prevent reports from being mailed to individual developers. The default setting is false.
report.mail.unknown	[email|user_name]	Specifies where to email reports for errors assigned to "unknown".
report.mail.on.error.only	true false	Enables/disables email reports to the manager when an error is found or a fatal exception occurs. Developer emails are not affected by this setting; developer emails are sent only to developers who are responsible for reported errors. The default setting is false.
report.setup.problems	top bottom hidden	Determines placement of setup problems section in report. The default setting is bottom.
report.setup.problems.category_limit	[numerical value]	Specifies a limit to the number of messages reported in a single setup problem category. Default is 10
report.setup.problems.display_limit	[numerical value]	Specifies a limit to the total number of messages displayed in the HTML report in the setup problem section. Default is 100
report.setup.problems.console	true false	Determines whether setup problems will be printed on the console. The default setting is true.
report.ue_coverage_details_htmls	LC DC	Specifies type of coverage included in an additional report, which includes source code annotated with line-by-line coverage details, when a test's HTML report links to it. LC: line coverage DC: decision coverage
report.separate_vm.xmx	[size]	Specifies how much memory should be used for reports generation. The default is 1024M.

Setting	Value	Description/Notes
report.separate_vm	true false	Enables/disables generating reports as a separate virtual machine. Default is false.
report.separate_vm.launch.file	[path]	Specifies path to launch file which should be used during reports generation.
dupcode.sorting.mode	oldest|newest|paths	Determines how elements in the code duplication findings are sorted. oldest: the oldest result appears at the top. newest: the newest result appears at the top. paths: sorts by full path names in ascending alphabetical order (A to Z). The default is paths.
report.coverage.version	1 2	Specifies the version of the XML coverage report: 1: the standard version will be used. 2: the size of the XML report will be optimized. The default value is 1.

General Source Control Settings

Setting Name	Value	Description/Notes
scontrol.timeout	[seconds]	Specifies timeout value for operations with source control. The default value is 60.
scontrol.branch	[name]	Enables you to specify a custom name for the tested branch. This setting may be used to describe unique runs. If it is not specified, the tested branch is detected automatically based on code provided to analysis.

AccuRev Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	accurev	AccuRev repository type identifier.

Setting Name	Value	Description/Notes
scontrol.accurev.exec	[path]	Path to external client executable (accurev).
scontrol.rep{n}.accurev.host	[host]	AccuRev server host.
scontrol.rep{n}.accurev.port	[port]	AccuRev server port. Default port is 1666.
scontrol.rep{n}.accurev.login	[login]	AccuRev user name.
scontrol.rep{n}.accurev.password	[password]	AccuRev password.

ClearCase Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	ccase	ClearCase repository type name.
scontrol.ccase.exec	[path]	Path to external client executable (cleartool).
scontrol.rep{n}.ccase.vob	[path]	Specifies the VOB's mount point - the path at which the VOB will be accessed by user. Examples: scontrol.rep.ccase.vob=X:\myvob scontrol.rep.ccase.vob=/vobs/myvob
scontrol.rep{n}.ccase.vob_tag	[tag]	The VOB's unique tag in the ClearCase network region.

CVS Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	cvs	CVS repository type identifier.
scontrol.rep{n}.cvs.root	[root]	Full CVSROOT value.

Setting Name	Value	Description/Notes
scontrol.rep{n}.cvs.pass	[password]	Plain or encoded password. The encoded password should match password in the .cvspass file. For CVS, use the value in .cvspass from within the user's home directory. For CVSNT, use the value store in the registry under HKEY_CURRENT_USER\Software\Cvsnt\cvspass The password is saved in the registry when you first log into the CVS repository from the command line using `cvs login`. To retrieve the password, go to the registry (using `regedit`) and look for the value under HKEY_CURRENT_USER->CVSNT> cvspass. This displays your entire login name (e.g., :pserver:exampleA@exampleB:/exampleC) and encrypted password value.
scontrol.rep{n}.cvs.useCustomSSHCredentials	true false	Enables/disables using the cvs login and password for EXT/SSH connections. Default is `false`.
scontrol.rep{n}.cvs.ext.server	[cvs]	Specifies which CVS application to start on the server side if connecting to a CVS server in EXT mode. Has the same meaning as the CVS_SERVER variable. Default is `cvs`.
scontrol.rep{n}.cvs.ssh.loginname	[login]	Specifies the login for SSH connections (if an external program can be used to provide the login).
scontrol.rep{n}.cvs.ssh.password	[password]	Specifies the password for SSH connection.
scontrol.rep{n}.cvs.ssh.keyfile	[file]	Specifies the private key file to establish an SSH connection with key authentication.
scontrol.rep{n}.cvs.ssh.passphrase	[passphrase]	Specifies the passphrase for SSH connections with the key authentication mechanism.
scontrol.rep{n}.cvs.useShell	true false	Enables/disables an external program (CVS_RSH) to establish a connection to the CVS repository. Default is `false`.

Setting Name	Value	Description/Notes
scontrol.rep{n}.cvs.ext.shell	[path]	Specifies the path to the executable to be used as the CVS_RSH program. Command line parameters should be specified in the cvs.ext.params property.
scontrol.rep{n}.cvs.ext.params	[parameters]	Specifies the parameters to be passed to an external program. The following case-sensitive macro definitions can be used to expand values into command line parameters: {host} repository host {port} port {user} cvs user {password} cvs password {extuser} parameter cvs.ssh.loginname {extpassword} parameter cvs.ssh.password {keyfile} parameter cvs.ssh.keyfile {passphrase} parameter cvs.ssh.passphrase

Git Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	git	Git repository type identifier.
scontrol.git.exec	[path]	Path to git executable. If not set, assumes git command is on the path.
scontrol.rep{n}.git.url	[url]	The remote repository URL (e.g., git://hostname/repo.git).
scontrol.rep{n}.git.workspace	[path]	The directory containing the local git repository.

Mercurial Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	hg	Mercurial reposity type identifyer.

Setting Name	Value	Description/Notes
scontrol.hg.exec	[path]	Path to external client executable. Devault is hg
scontrol.rep{n}.hg.url	[url]	The remote repository URL (e.g., http://hostname/path).
scontrol.rep{n}.hg.workspace	[path]	The directory containing the local Mercurial repository.

Perforce Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	perforce	Perforce repository type identifier.
scontrol.perforce.exec	[path]	Path to external client executable (p4).
scontrol.rep{n}.perforce.host	[host]	Perforce server host.
scontrol.rep{n}.perforce.port	[port]	Perforce server port. Default port is 1666.
scontrol.rep{n}.perforce.login	[login]	Perforce user name.
scontrol.rep{n}.perforce.password	[password]	Perforce password, optional if ticket is used for authentication.
scontrol.rep{n}.perforce.client	[client]	The client workspace name as specified in the P4CLIENT environment variable or its equivalents. Root directory for specified workspace should be configured correctly for local machine.

Serena Dimensions Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	serena	Serena Dimensions repository type identifier.
scontrol.serena.dmroot	[path]	Path to the Serena Dimensions executable. Example: C\:\\Program Files (x86)\\Serena\\Dimensions 2009 R2\\CM\\
scontrol.rep{n}.serena.login	[login]	Serena user name.
scontrol.rep{n}.serena.password	[password]	Password.
scontrol.rep{n}.serena.host	[host]	Serena Dimensions server host name.

Setting Name	Value	Description/Notes
scontrol.rep{n}.serena.dbname	[name]	Name of the database for the product you are working with.
scontrol.rep{n}.serena.dbconn	[connection]	Connection string for that database.
scontrol.rep{n}.serena.locale	[locale]	The language used, (e.g., en_US)
scontrol.rep{n}.serena.mapping	[mapping]	If the project has been downloaded/ moved to a location other than default work area, use this option to specify a mapping between the project or stream with the Serena repository and the local project. If you are working in the default work area, you do not need to define mappings.

StarTeam Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	starteam	StarTeam repository type identifier.
scontrol.rep{n}.starteam.host	[host]	StarTeam server host.
sscontrol.rep{n}.starteam.port	[port]	StarTeam server port. Default port is 49201.
scontrol.rep{n}.starteam.login	[login]	Login name.
scontrol.rep{n}.starteam.password	[password]	Password (not encoded).
scontrol.rep{n}.starteam.path	[path]	Specifies the project, view, or folder that you are currently working with. You can specify a project name (all views will be scanned when searching for the repository path), project/view (only the given view will scanned) or project/view/ folder (only the specified Star Team folder will be scanned). This setting is useful for working with large multi-project repositories. Examples: scontrol.rep.starteam.path=proj1 scontrol.rep.starteam.path=proj1/view1 scontrol.rep.starteam.path=proj1/ view1/folderA scontrol.rep.starteam.path=proj1/ view1/folderA/folderB

Setting Name	Value	Description/Notes
scontrol.rep{n}.starteam.workdir	[path]	Specifies a new working directory for the selected view's root folder (if the path represents a view) or a new working directory for the selected folder (if the path represents a folder) when the scontrol.rep.starteam.path setting points to a StarTeam view or folder. Examples: scontrol.rep.starteam.workdir=C:\\storage\\dv scontrol.rep.starteam.workdir=/home/storage/dv

Subversion Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	svn	Subversion repository type identifier.
scontrol.svn.exec	[path]	Path to external client executable (svn).
scontrol.rep{n}.svn.url	[url]	Subversion URL specifies protocol, server name, port and starting repository path. Example: svn://buildmachine.foobar.com/home/svn
scontrol.rep{n}.svn.login	[login]	Login name.
scontrol.rep{n}.svn.password	[password]	Password (not encoded).

Synergy/CM Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	synergy	Synergy/CM repository type identifier.
scontrol.synergy.exec	[path]	Path to external client executable (ccm).
scontrol.rep{n}.synergy.host	[host]	Computer on which synergy/cm engine runs. Local host is used when missing. For Web mode, the host must be a valid Synergy Web URL with protocol and port (e.g., http://synergy.server:8400).
scontrol.rep{n}.synergy.dbpath	[path]	Absolute synergy database path (e.g., \\host\db\name).

Setting Name	Value	Description/Notes
scontrol.rep{n}.synergy.projspec	[specification]	Synergy project specification which contains project name and its version (e.g., name-version).
scontrol.rep{n}.synergy.login	[login]	Synergy user name.
scontrol.rep{n}.synergy.password	[password]	Synergy password (not encoded).
scontrol.rep{n}.synergy.port	[port]	Synergy port.
scontrol.rep{n}.synergy.remote_client	[client]	(UNIX only) Specifies that you want to start ccm as a remote client. Default value is false. Optional. This is not used for Web mode.
scontrol.rep{n}.synergy.local_dbpath	[path]	Specifies the path name to which your database information is copied when you are running a remote client session. If null, then the default location will be used. This is not used for Web mode.

Microsoft Team Foundation Server Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	tfs	TFS repository type identifier.
scontrol.rep{n}.tfs.url	[url]	URL to TFS repository, e.g., http://localhost:8080/tfs
scontrol.rep{n}.tfs.login	[login]	TFS user name.
scontrol.rep{n}.tfs.password	[password]	TFS password.

Microsoft Visual SourceSafe Source Control Settings

Setting Name	Value	Description/Notes
scontrol.rep{n}.type	vss	Visual SourceSafe repository type identifier.
scontrol.vss.exec	[path]	Path to external client executable (ss).
scontrol.rep{n}.vss.ssdir	[path]	Path of repository database.
scontrol.rep{n}.vss.projpath	[path]	VSS project path.
scontrol.rep{n}.vss.login	[login]	VSS login.
scontrol.rep{n}.vss.password	[password]	VSS password.

Other Settings

Setting Name	Value	Description/Notes
`jtest.fail`	`true` `false`	Fails the build by returning a non-zero exit code if any violation is reported
`jtest.show.settings`	`true` `false`	Prints the current settings and customizations, showing the source of each configuration entry (for example, .properties file)
`jtest.data.additional`	`[path]`	Path to additional json.data file. In case of may files, use a comma separated list of file paths

Integrations

- Integrating with Source Control Systems
- Build Systems Integration
- External Analyzers Integration
- Using DTP Engines in an IDE
- Integrating with CI Tools

Build Systems Integration

Jtestcli runs on `jtest.data.json` files created by the build system plug-ins. See “Specifying Test Data Location”, page 11, for information about the structure of the `jtest.data.json` file. For details on plug-ins, including troubleshooting issues with Maven, Ant, or Gradle, see manuals/plugins-manual.html.

Additional Information for Jtest 9.5 Users

The release of Static Analysis Engine for Java marks a substantial improvement over Jtest 9.x support for build automation systems. Static Analysis Engine integrates with builds differently than previous Jtest 9.x releases. Static Analysis Engine retains Jtest abilities, but the current release no longer needs to leverage Eclipse and import projects into workspace to perform analysis. This change significantly reduces execution time.

The command line tool has also been modified. As a result, part of build system plugins' options have been changed in order to preserve naming consistency.

External Analyzers Integration

Checkstyle

DTP Engines supports static analysis with Checkstyle 5.5, 5.6, 5.7, and 6.5.

To enable Checkstyle rules and to use plugins dropped into this folder, you need to reconfigure the settings in the $JTEST_INSTALL_DIR/etc/framework.properties configuration file.

Add the following .jar file as the first entry to the `felix.auto.start.4` set:

`org.apache.felix.fileinstall-3.4.0.jar`

Downloading and Installing Checkstyle

1. Download Checkstyle from the following website: http://sourceforge.net/projects/checkstyle/files
2. Unpack Checkstyle to install.
3. Copy the configs, plugins, and rules folders from the $HOME/integration/checkstyle directory into the installation directory
4. Modify the jtestcli.properties file to include the following properties:

```
jtest.analyzer.checkstyle.enabled=true
jtest.analyzer.checkstyle.dir=CHECKSTYLE/INSTALLATION/DIRECTORY
rules.provider_cs.analyzer=com.puppycrawl.tools.checkstyle
rules.provider_cs.data=${jtest.home}/rules/csrules.xml
```

Analyzing Code with Checkstyle

Specify the Checkstyle user test configurations to perform analysis with Checkstyle.

`user://Checkstyle`	Enables every rule
`user://Checkstyle Recommended`	Enables Checkstyles recommended rules

Optional Customizations

You can customzie your Checkstyle integration by configuring the following properties.

```
# Custom limit for number of files to be tested in a single executed Checkstyle process. Default is 50.
# However, if subsequent files are placed in the same directory, they will be still scheduled to that
# process unless the "hard" limit is reached (twice the "regular" limit).
jtest.analyzer.checkstyle.files.limit=50

# Custom timeout (seconds) for the Checkstyle process to start. Default is 5 seconds.
jtest.analyzer.checkstyle.launch.timeout=5

# Custom timeout (seconds) for the Checkstyle process to complete. Default is 60 seconds.
jtest.analyzer.checkstyle.timeout=60

# Custom launch file for Checkstyle process. Default is taken from the Checkstyle analyzer jar file.
# If one cannot be found, attempts to use file 'checkstyle.ini' from current working directory.
jtest.analyzer.checkstyle.launch.file=CUSTOM/LAUNCH/FILE
```

The rulemap.txt file located in the $HOME/integration/checkstyle directory contains information about how rule IDs used in test configuration are mapped to Checkstyle rule types.

FindBugs

Static Analysis Engine supports flow analysis with FindBugs 2.0.2, 2.0.3, 3.0 and 3.0.1.

Downloading and Installing FindBugs

1. Download FindBugs from the following website: http://findbugs.sourceforge.net/downloads.html
2. Unpack FindBugs to install.
3. Copy the configs, plugins, and rules folders from the $HOME/integration/findbugs directory into the installation directory
4. Modify the jtestcli.properties file to include the following properties:

```
jtest.analyzer.findbugs.enabled=true
jtest.analyzer.findbugs.dir=FINDBUGS/INSTALLATION/DIRECTORY
rules.provider_fb.analyzer=edu.umd.cs.findbugs
rules.provider_fb.data=${jtest.home}/rules/fbrules.xml
```

Analyzing Code with FindBugs

Specify the FindBugs user test configurations to perform analysis with Findbugs.

`user://FindBugs`	Enables every rule
`user://FindBugs Recommended`	Enables FindBugs recommended rules

Project must be built prior to analysis because Findbugs requires compiled classes to be present/

Optional Customizations

You can customzie your FindBugs integration by configuring the following properties.

```
# Custom limit for number of files to be tested in a single executed FindBugs process.
Default is reasonably high.
jtest.analyzer.findbugs.files.limit=1000

# Custom timeout (seconds) for the FindBugs process to start. Default is 5 seconds. /
jtest.analyzer.findbugs.launch.timeout=5

# Custom timeout (seconds) for the FindBugs process to complete. Default is 360 seconds. /
jtest.analyzer.findbugs.timeout=360

# Custom launch file for FindBugs process. Default is taken from the FindBugs analyzer jar
file.
# If one cannot be found, attempts to use file 'findbugs.ini' from current working directory.
jtest.analyzer.findbugs.launch.file=CUSTOM/LAUNCH/FILE
```

The rulemap.txt file located in the $HOME/integration/findbugs directory contains information about how rule IDs used in test configuration are mapped to FindBug rule types.

Integrating with Source Control Systems

DTP Engines can collect information from source control systems and use the data to assign ownership of violations, filter analyzed files based on time or modification history, and report information about controlled files to DTP Server. Use the Jtest 9.5 or later interface to configure integration with source control systems:

1. In your IDE , choose **Parasoft > Preferences** and click **Source Controls**
2. Configure your repository and source control client and click **Apply**.
3. In the Preferences panel menu, click **Scope and Authorship**
4. Enable the **Use source control (modification author) to compute scope** option and click **Apply**.
5. In the Preferences panel menu, click Parasoft
6. Click the **share** to open the Export to localsettings file panel.
7. Select the **Source Controls**, **Scope and Authorship**, and any other options you want to save.
8. Choose a location and click **OK**.
9. Add the following line to the settings file, which ensures that information on source control details are saved to the report:

```
report.scontrol=min
```

10. Either pass the file to the command line or copy the settings in the administration panel of a project in DTP server (Parasoft Test settings tab) if applicable.
11. Run the analysis.

Integrating with CI Tools

Integrating with Jenkins

DTP Engines for Java can be integrated with Jenkins continuous integration software. The Parasoft Findings Plugin for Jenkins allows you to visualize static analysis and test results as trend graphs and warnings.

Parasoft Findings Plugin is available directly in Jenkins. See Parasoft Findings Plugin for details.

You can download the plugin source files form GitHub, see Parasoft Findings Plugin Project. If you need additional information on how to rebuild the plugin, contact Parasoft Support.

Getting Help

Use the the -help switch to access usage information on the command line.

```
jtestcli.exe -help
```

Technical Support

You can configure DTP Engines to create package for technical support. Add the following settings to your .properties configuration file:

```
techsupport.enabled=true
techsupport.create.on.exit=true
techsupport.archive.location=[OUTPUT DIRECTORY]
```

A technical support package will be created in the output directory at the end of an analysis run.

You need to escape the backslashes to specify the package location. The following example shows what the path may look like:

```
C:\\Project\\Mailsystem\\Report
```

Troubleshooting

Floating Machine ID

Changes in the network environment may affect the interface that is used to compute your machine ID and result in machine ID instability. You can use the PARASOFT_SUPPORT_NET_INTERFACES environment variable to specify a stable interface and prevent the machine ID from floating.

1. Set up the **PARASOFT_SUPPORT_NET_INTERFACES** environment variable.
2. Set the variable value to a stable Ethernet network interface. Do not use virtual, temporary or loopback interfaces.
    - On Windows: Set the value to the MAC address of your network card. You can use the `ipconfig -all` command to obtain the address. Example:

```
SET PARASOFT_SUPPORT_NET_INTERFACES=00-10-D9-27-AC-85
```

    - On Linux: Set the value to one of the network interfaces from the "inet" or "inet6" family. For example: You can use the `ifconfig` command to obtain the list of available interfaces. Example:

```
export PARASOFT_SUPPORT_NET_INTERFACES=eth1
```

If the problem persists, you can obtain diagnostic information by setting up the environment variable **PARASOFT_DEBUG_NET_INTERFACES** and setting its value to `true`. This will print to the standard output the checking procedure that can be shared with technical support , as well as the inter-

face that is used to compute your machine ID. The interface will be marked with the `[SELECTED]` prefix.

Third-Party Content

DTP Engines for Java incorporate items that have been sourced from third parties. The names of the items and their license agreements have been listed in the table. Click the license name to see the details.

Item	License
commons-collections.jar	Apache License 2.0
commons-vfs.jar	Apache License 2.0
avalon-framework.jar	Apache License 2.0
batik-all.jar	Apache License 2.0
fop.jar	Apache License 2.0
chardet.jar	Mozilla Public License
bcprov.jar	MIT License
saxon.jar	Mozilla Public License
jfreechart.jar	GNU LGPL License
jcommon.jar	GNU LGPL License
cvslib.jar	CDDL License
javax.xml.stream_1.0.1.jar	Eclipse Public License
javax.activation_1.1.1.jar	Apache License 2.0
jakarta-log4j.jar	Apache License 2.0
xmlgraphics-commons.jar	Apache License 2.0
fst.jar	Apache License 2.0
truezip.jar	Apache License 2.0
jjawin.jar	DevelopMentor OpenSource Software License
trilead-ssh2.jar	Trilead AG License
javanet.staxutils_1.0.0.jar	BSD License
commons-codec.jar	Apache License 2.0
commons-httpclient.jar	Apache License 2.0
org.apache.commons.io_1.4.0.v20081110-1000.jar	Apache License 2.0
org.apache.commons.logging_1.1.3.jar	Apache License 2.0
httpclient.jar	Apache License 2.0
httpcore.jar	Apache License 2.0

Item	License
httpclient.jar	Apache License 2.0
httpmime.jar	Apache License 2.0
org.apache.jcs_1.3.4.jar	Apache License 2.0
org.codehaus.stax2_3.2.4.jar	Apache License 2.0
org.json_1.0.0.v201507290100.jar	JSON License
javax.mail_1.5.0.jar	CDDL License
org.suigeneris.jrcs.diff_0.4.2.jar	GNU LGPL License
org.apache.felix.scr-1.6.2.jar	Apache License 2.0
osgi.core-5.0.0.jar	Apache License 2.0
antlr-2.7.7.jar	ANTLR 4 License (BSD)
asm	ASMDEX License
asm-analysis	ASMDEX License
asm-commons	ASMDEX License
asm-tree	ASMDEX License
asm-util	ASMDEX License
c3p0-0.9.5.jar	GNU LGPL License 2.1, Eclipse Public License 1.0
com.esotericsoftware.minlog_1.2.0.jar	BSD License
commons-collections-3.2.2.jar	Apache License 2.0
commons-compress-1.11.jar	Apache License 2.0
commons-configuration_1.9.0.jar	Apache License 2.0
commons-dbutils-1.6.jar	Apache License 2.0
commons-exec-1.1.jar	Apache License 2.0
commons-io-2.2.jar	Apache License 2.0
commons-io_2.4.0.jar	Apache License 2.0
commons-lang-2.6.jar	Apache License 2.0
commons-lang3-3.1.jar	Apache License 2.0
com.sun.xml.bind.2.2.7.jar	CDDL and LGPL 2.1
dom4j-1.6.1.jar	BSD License
doxia-core-1.2.jar	Apache License 2.0
doxia-decoration-model-1.2.jar	Apache License 2.0

Item	License
doxia-logging-api-1.2.jar	Apache License 2.0
doxia-module-xhtml-1.2.jar	Apache License 2.0
doxia-sink-api-1.2.jar	Apache License 2.0
doxia-site-renderer-1.2.jar	Apache License 2.0
gson-2.2.4.jar	Apache License 2.0
guava_18.0.0.jar	Apache License 2.0
h2-1.4.186.jar	Mozilla Public License 2.0, or Eclipse Public License 1.0
hibernate-c3p0-4.3.5.Final.jar	GNU LGPL License 2.1
hibernate-commons-annotations-4.0.4.Final.jar	GNU LGPL License 2.1
hibernate-core-4.3.5.Final.jar	GNU LGPL License 2.1
hibernate-jpa-2.1-api-1.0.0.Final.jar	Eclipse Public License 1.0, Eclipse Distribution License 1.0
jandex-1.1.0.Final.jar	Apache License 2.0
javassist-3.18.1-GA.jar	Mozilla Public License 1.1, LGPL 2.1, Apache License 2.0
javassist-3.18.2-GA.jar	Mozilla Public License 1.1, LGPL 2.1, Apache License 2.0
jaxb-api_2.2.7.jar	CDDL 1.1,GPL 2
jboss-logging-3.1.3.GA.jar	Apache License 2.0
jboss-transaction-api_1.2_spec-1.0.0.Final.jar	CDDL, GPL 2.0 with the Class-path Exception
jsr305-2.0.3.jar	Apache License 2.0
junit-4.11.jar	Common Public License 1.0
jython_2.5.3.jar	Jython Software License
kryo_2.24.0.jar	BSD License
lucene-core.jar	Apache License 2.0
maven-artifact-2.1.0.jar	Apache License 2.0
maven-reporting-api-3.0.jar	Apache License 2.0
maven-reporting-impl-2.2.jar	Apache License 2.0
mchange-commons-java-0.2.9.jar	GNU LGPL License 2.1, Eclipse Public License 1.0

Item	License
mockito-all-1.9.5.jar	MIT License
objenesis-2.1.jar	Apache License 2.0
org.abego.treelayout.core_1.0.1.jar	BSD 3-Clause 'New' or 'Revised' License (BSD-3-Clause)
org.antlr.4-annotations_4.2.1.jar	ANTLR 4 License (BSD)
org.antlr.4-runtime_4.2.1.jar	ANTLR 4 License (BSD)
org.apache.felix.fileinstall-3.5.0.jar	Apache License 2.0
org.apache.felix.gogo.command-0.14.0.jar	Apache License 2.0
org.apache.felix.gogo.runtime-0.12.1.jar	Apache License 2.0
org.apache.felix.gogo.shell-0.10.0.jar	Apache License 2.0
org.apache.felix.main-4.4.1.jar	Apache License 2.0
org.apache.felix.scr-1.8.2.jar	Apache License 2.0
org.eclipse.core.contenttype_3.5.100.jar	Eclipse Public License 1.0
org.eclipse.core.filesystem_1.6.0.jar	Eclipse Public License 1.0
org.eclipse.core.jobs_3.8.0.jar	Eclipse Public License 1.0
org.eclipse.core.resources_3.11.0.jar	Eclipse Public License 1.0
org.eclipse.core.runtime_3.12.0.jar	Eclipse Public License 1.0
org.eclipse.equinox.common_3.8.0.jar	Eclipse Public License 1.0
org.eclipse.equinox.preferences_3.6.0.jar	Eclipse Public License 1.0
org.eclipse.jdt.core_3.12.0.jar	Apache License 2.0
org.eclipse.osgi_3.11.0.jar	Eclipse Public License 1.0
org.eclipse.text_3.6.0.jar	Eclipse Public License 1.0
org.objenesis_2.1.0.jar	Apache License 2.0
org.restlet	Apache License 2.0
org.restlet.ext.json	Apache License 2.0
org.restlet.ext.simple	Apache License 2.0
org.restlet.lib.org.json	Apache License 2.0
org.simpleframework	Apache License 2.0
oro_2.0.8.jar	Apache License 1.1

Item	License
plexus-archiver-2.10.1.jar	Apache License 2.0
plexus-i18n-1.0-beta-10.jar	Apache License 2.0
plexus-io-2.6.jar	Apache License 2.0
plexus-utils-2.1.jar	Apache License 2.0
plexus-velocity-1.1.8.jar	Apache License 2.0
powermock-mockito-1.5.5-full.jar	Apache License 2.0
servlet-api.jar	Apache License 2.0
spell.jar	Apache License 2.0
stax2-api-3.1.1.jar	BSD License
velocity-1.7.jar	Apache License 2.0
woodstox-core-lgpl-4.2.0.jar	GNU LGPL License 2.1
xmlpull_1.1.3.1.jar	Public Domain
xpp3.min_1.1.0.4c.jar	Public Domain
xstream_1.4.4.jar	BSD style