



Development Testing Platform Engines for .NET 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 off-site 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 agree 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.** IN 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 purchase 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

Introduction

Static Analysis Engine (SAE)	5
Unit Test Connector (UTC)	5
Code Coverage Engine (CCE)	5

Getting Started

System Requirements	6
Installing DTP Engines	7
Setting the License	9
Connecting to DTP Server	10
Connecting to Source Control	11
Known Limitations	11

Static Analysis Engine

Basic Analysis	13
Analyzing Visual Studio Solutions	13
Analyzing Visual Studio Projects Without Solutions	13
Analyzing Websites	13
Specifying Test Configurations	15
Viewing Available Test Configurations	15
Built-in Test Configurations	15
Creating Custom Rules	17
Defining Test Scope	18
Testing a Single Project in a Solution	18
Testing a Single Directory of Files in a Project	18
Testing a Single Source File	18
Testing a Single Project Under a Solution Folder	18
Testing a Single Source File When No Solution is Provided	18
Fine-tuning the Scope	18
Specifying Additional Assemblies	19

Configuring Authorship	20
About Authorship Configuration Priority	20
Configuring How Authorship is Computed	20
Creating Authorship XML Map Files	21
Suppressing Violations	22
Line Suppression	22
Flow Analysis.....	24
Configuring Depth of Flow Analysis	24
Setting Timeout Strategy	25
Running Flow Analysis with Swapping of Analysis Data Enabled	25
Configuring Verbosity of Flow Analysis	25
Null-checking Methods	26
Specifying Resources	27
Metrics Analysis.....	29
Setting Metrics Thresholds	29
Code Duplicate Analysis.....	30
Using DTP Engines in an IDE	31

Reporting

Specifying Report Output Location	32
Specifying Report Format	32
Viewing Reports	32
Sending Results to Development Testing Platform (DTP) Server	38
Publishing Source Code to DTP Server	38

Unit Test Connector

Running Unit Tests with Coverage	39
Running NUnit Tests	39
Running MSTest Tests	39
Associating Tests with Development Artifacts	40

Code Coverage Engine

Application Coverage for Standalone Applications.....	44
Merging Coverage Data	46
Application Coverage for Web Applications	47
Prerequisites	47
Process Overview	47
Configuring the Application Under Test for Coverage	47
Test Configuration and Execution	50
Uploading Test Results to DTP	50
Generating a Dynamic Coverage Data File and Uploading it to DTP	50
Stopping Dynamic Coverage Data Collection	50
Known Limitations	51
Reviewing Coverage in DTP	51

Customizing DTP Engines for .NET

Auto-configuring Settings from DTP Server	52
Using Variables	53
Settings Reference	54

Integrations

Integrating with MSBuild.....	79
Target File	79
Integrating with NAnt	81
Loading NAnt Task Library	81
Integrating with Source Control Systems	85
Integrating with CI Tools.....	86
Integrating with TeamCity	86
Integrating with Jenkins	86

Building Solutions and Projects

Delegating the Build to MSBuild or Visual Studio	88
Depending on Pre-built Code	88
Specifying Solution Configuration and Target Platform	88

Verifying the Required Build Artifacts	88
--	----

Getting Help

Technical Support	89
Troubleshooting	89
Switches Reference	91

Third-Party Content

Introduction

Parasoft Development Testing Platform (DTP) Engines for .NET 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 C# and VB.NET applications function as expected. See "Static Analysis Engine", page 12.

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 39.

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 43.

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 10, Windows 8, Windows 7, Windows Server 2012, or Windows Server 2008
- 4GB memory minimum, 8GB recommended
- 2GHz or faster processor
- .NET Framework 4.6, 4.5, 4.0, or 3.5 installed on the target machine. See “About .NET Framework Prerequisite”, page 6.
- Microsoft Visual C++ Redistributable Packages. If not already present, the packages are automatically installed during DTP Engines for .NET installation. Also see “Manually Installing Visual C++ Redistribution Packages”, page 6.

About .NET Framework Prerequisite

.NET Framework 3.0 and older are not sufficient for DTP Engines. Version 4.5 of .NET Framework is bundled with DTP Engines and installed automatically. To install the framework manually, run the following command:

```
[INSTALL_DIR]\bin\prerequisites\dotnetfx45_full_x86_x64.exe
```

You can download and install other supported versions from Microsoft. The following table describes .NET Framework version support.

Framework Version	Static Analysis	Unit Testing	Metrics
4.6	Supported	Supported	Supported
4.5	Supported	Supported	Supported
4.0	Supported	Supported	Unsupported
3.5	Supported	Supported	Unsupported

Manually Installing Visual C++ Redistribution Packages

If you installed DTP Engines from a ZIP distribution, the Visual C++ Redistribution Packages necessary for full functionality may not have been installed. You can install them manually by running the following installers located in the `[INSTALL_DIR]\bin\prerequisites\` directory:

- `vcredist_x64_10.exe`
- `vcredist_x64_9.exe`
- `vcredist_x86_10.exe`
- `vcredist_x86_9.exe`

Installing DTP Engines

Running the installation executable launches a graphical interface that simplifies installing or updating DTP Engines for .NET. Using this method ensures that all prerequisites are installed and that permissions are elevated to work with User Access Control. Running the installer executable also allows you to use the Windows Control Panel to uninstall DTP Engines.

1. Run `parasoft_dottest_[version].exe` setup file
2. Follow the steps as shown in the installation wizard.

Installing from a Zip Distribution

DTP Engines are also available for installation as a .zip file that you can manually deploy to your desired directory. If you are installing a customized version of DTP Engines, e.g., contains a modified set of built-in test configurations, then you may need to install from the .zip file.

We recommend extracting the contents of the .zip file with unzip software other than the built-in Windows unzip utility, which may consider the downloaded .zip file to be untrusted.

The following components must be installed manually prior to installing DTP Engines using this method:

- NET Framework 4.0 or .NET Framework 3.5
- VC++ Redistributable Package x86

These components ship with DTP Engines in the `[INSTALL_DIR]\bin\prerequisites` directory.

Connecting to Build and Continuous Integration Systems

DTP Engines ship with a set of integrators that allow you to easily integrate with the following build and continuous integration systems:

- MSBuild
- NAnt
- JetBrains TeamCity

When the "Select Build System for Integration" tab appears during installation, choose the appropriate build systems and follow the deployment wizards. For instructions on how to integrate with your build or continuous integration system, see the appropriate section in the "Integrations", page 78, chapter.

You can deploy the integrators after DTP Engines have been installed by running the following command and following the wizard instructions:

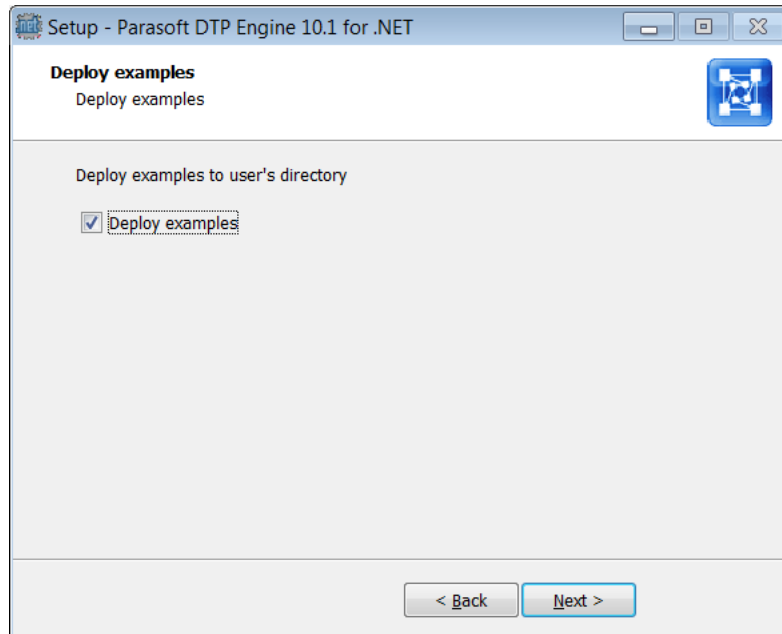
```
[INSTALL_DIR]\Integrations\[Build System]\deploy.exe
```

Deploying Examples

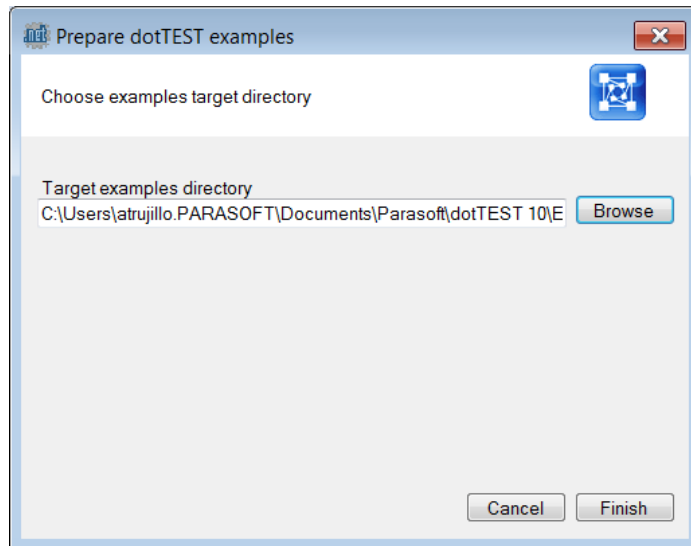
DTP Engines for .NET ships with several examples to help you become familiar with its functionalities. You can deploy the examples during installation or on demand after the installation has been completed.

Deploying Examples During Installation

1. Enable the Deploy examples option when prompted and click **Next**



2. Choose a location for the examples and click **Finish**. If the examples directory already exists, the previous files will be moved to the recycle bin and replaced with a clean image of the example files.



After the examples are deployed, the examples directory opens so that you can browse the content. Examples are organized into directories for all supported versions of Visual Studio. Each directory contains subdirectories covering various functionality. Follow the instructions in the ReadMe file in each directory for tutorials on using DTP Engines for .NET. We recommend starting with the 010_Basics examples.

The examples point to the instance of DTP Engines from which they were deployed. This allows you to run the examples with minimal configuration effort. The license, however, must be configured to start using the examples. See "Connecting to DTP Server", page 10.

Some files contain absolute paths to where they were deployed, so moving the example files will cause them to fail. Redeploy the examples to a new location instead of moving them if necessary.

Deploying Examples from the Command Line

Run the following command to deploy the examples shipped with DTP Engines:

```
[INSTALL_DIR]\deploy_examples.exe
```

Follow the same processes as described in “Deploying Examples During Installation”, page 7.

Multiple Installations on a Single Machine

Run the installer in extract mode to install DTP Engines two or more times on a single machine.

```
parasoft_dottest_10.X.Y.exe /extract
```

Running dotTEST with a Local System Account

Using dotTEST from a Local System account may impact some functionalities, such as integration with MSTest and collecting coverage information for applications or NUnit tests. To ensure that dotTEST functions as expected, you may need to manually run `Parasoft.Dottest.Profiling.Reg.exe` located in `<INSTALL_DIR>\bin\dottest\dotnet`. Add the `-ForAllUsers` switch to the executable:

```
<INSTALL_DIR>\bin\dottest\dotnet\parasoft.dottest.profiling.reg.exe -ForAllUsers
```

Ensure you have admin credentials before you run the executable.

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 `dottest.license.use_network` property to `false`
2. Set the `dottest.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:

```
dottestcli -machineID
```

Network License

In the `.properties` configuration file:

1. Set the `dottest.license.use_network` property to `true`
2. Set the `dottest.license.network.type`
3. Set the `dottest.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 38)
- Sharing test configurations
- Sharing static analysis rules

Modify the following settings in the `[INSTALL_DIR]\dottestcli.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 `dottestcli.properties` file.

```
ntp.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
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 `dottestcli.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 .NET”, page 52, for information about configuring your SCM connection.

If you have dotTEST 9.5 or later, you can use its interface to configure integration with source control systems. See “Integrating with Source Control Systems”, page 85 for details.

Known Limitations

Please refer to the Known Limitations section in `[INSTALL_DIR] \releasenotes.rtf` for information.

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
- Metrics Analysis
- Code Duplicate Analysis
- Using DTP Engines in an IDE

Basic Analysis

The command-line is the primary interface for running Static Analysis Engine in non-interactive mode. The interface allows you to run Static Analysis Engine from any other script, language or build system that can launch a program, including *.bat and *.cmd scripts. For MSBuild, NAnt, or JetBrains TeamCity, use the integrators shipped with DTP Engines. See "Integrations", page 78, for details.

Analyzing Visual Studio Solutions

Provide the paths of a solution to analyze code. The following versions of Visual Studio solutions can be analyzed: 2005, 2008, 2010, 2012, 2013, 2015.

Analyzing a Single FileSolution

Use the `-solution` switch to specify path of the solution, e.g.:

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-config "builtin://Demo" -report "C:\Report"
```

Analyzing Multiple Solutions

Specify `-solution` multiple times or provide ANT-style wildcards, e.g.:

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-solution "C:\Devel\BarSolution\BarSolution.sln"
-config "builtin://Demo" -report "C:\Report"
```

or

```
dottestcli.exe -solution "C:\Devel\**\*.sln" -config
"builtin://Demo" -report "C:\Report"
```

Analyzing Visual Studio Projects Without Solutions

You can specify a path or paths to *.csproj or *.vbproj projects if *.sln files are unavailable by using the `-project` switch. The `-solution` switch is recommended, however, and should be used whenever possible.

```
dottestcli.exe -project "C:\Devel\FooSolution\**\*.csproj"
-config "builtin://Demo" -report "C:\Report"
```

ANT-style paths are supported. The switch may be specified several times.

Analyzing Websites

Visual Studio Web Site projects do not use *.proj files and may be maintained without *.sln files. Use the -website switch to specify the directory of the website when no *.sln files are present, e.g.:

```
dottestcli.exe -website "C:\Devel\FooWebSite" -config "builtin://Demo"  
-report "C:\Report"
```

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:

```
dottestcli.exe -solution "C:\Devel\MyFooSolution\MySolution.sln"
-config "builtin://Demo" -report "C:\Report"
```

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.

```
-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.
Metrics	Computes values for several code metrics.
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.
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	Includes rules for demonstrating various techniques of code analysis. May not be suitable for large code bases.
Find Memory Issues	Includes rules for finding memory management issues in the code.
Find Unimplemented Scenarios	Includes rules for finding unimplemented scenarios in the code.
Find Unused Code	Includes rules for identifying unused/dead code.
Check Code Compatibility against .NET [2.0, 3.0, 3.5, 4.0 Client Profile, 4.0 Full]	Includes a set of test configurations that validates the code's compatibility with the specified version of .NET framework.
CWE-SANS Top 25 Most Dangerous Programming Errors	Includes rules that find issues classified as Top 25 Most Dangerous Programming Errors of the CWE-SANS standard.

Configuration Name	Description
NIST SAMATE	Includes rules that find issues identified in the NIST SAMATE standard
OWASP Top 10 Security Vulnerabilities	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
Security Assessment	General test configuration that finds security issues
IEC 62304 (Template)	A template test configuration for applying the IEC 62304 Medical standard.
Run NUnit Tests	Runs NUnit Tests that are found in the scope of analysis
Run NUnit Tests with coverage	Runs NUnit Tests that are found in the scope of analysis and monitors coverage
Calculate Application Coverage	Processes the application coverage data to generate a coverage.xml file. See "Application Coverage for Web Applications", page 47 for additional information.
Execute MSTests with Coverage	Executes MSTests and collects coverage. See "Running MSTest Tests", page 39, for more information.
Execute MSTests	Executes MSTests. See "Running MSTest Tests", page 39, for more information.
Collect Static Coverage	Generates the static coverage data necessary for application coverage. See "Application Coverage for Web Applications", page 47, for details.

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

The test scope refers to the file or set of files for testing. Use the `-resource` switch followed by a path in the solution to define the scope. Do not use file system paths to define the scope. Use the Visual Studio Solution Explorer path instead.

If you are running analysis from your IDE, a source file that is open in the active editor has higher priority than resources defined with Solution Explorer and only this file will be analyzed.

Testing a Single Project in a Solution

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-resource "FooSolution/QuxProject"
-config "builtin://Demo" -report "C:\Report"
```

Testing a Single Directory of Files in a Project

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-resource "FooSolution/BarProject/QuxDirectory"
-config "builtin://Demo"
```

Testing a Single Source File

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-resource "FooSolution/BarProject/QuxDirectory/BazFile.cs"
-config "builtin://Demo"
```

Testing a Single Project Under a Solution Folder

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-resource "FooSolution/BarSolutionFolder/QuxProject"
-config "builtin://Demo" -report "C:\Report"
```

Testing a Single Source File When No Solution is Provided

Because the name of the solution is unknown, the solution path should start from `/`.

```
dottestcli.exe -project "C:\Devel\FooSolution\FooProject.csproj"
-resource "/FooProject/BarDirectory/QuxFile.cs"
-config "builtin://Demo" -report "C:\Report"
```

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.

These switches accept file system paths and ANT-style wildcards. This is in contrast to the `-resource` switch, which accepts the solution path and ANT-style wildcards.

The following example shows how to exclude all files under directories `*.Tests:`

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-exclude "C:\Devel\FooSolution\*.Tests\**\*.*"
-config "builtin://Demo" -report "C:\Report"
```

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.

Specifying Additional Assemblies

Use the `-reference` switch to specify a path to additional assemblies needed to resolve dependencies of the analyzed projects. ANT-style wildcards and relative paths to the current working directory are accepted.

Examples

```
-reference C:\MySolution\ExternalAssemblies\*.dll
-reference C:\MySolution\ExternalAssemblies\*.exe
-reference C:\MySolution\ExternalAssemblies\**\*.dll
-reference C:\MySolution\ExternalAssemblies\**\*.dll
```

Use the `-reference` switch if you receive an "Unable to find reference assembly" message.

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.

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. current user

If one of the selected options does not determine an author, 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 `dottestcli.properties` configuration file to specify how authorship is determined:

```
scope.local=[true or false]
scope.scontrol=[true or false]
scope.xmlmap=[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[identifier]` 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/Foo.c	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"

```


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.

You can enable or disable the mode by using the test configuration interface in DTP:

Enable swapping of analysis data to disk: Disabled 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 regu-

lar 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 `Account` and `AccountManager` in the following example are defined in a different assembly and are outside the scope of the analysis. The class `AccountManager` uses static methods to check `Account` in following way:

```
public class Account
{
    public int Balance { get; set; }
}

public class AccountManager
{
    public static bool IsNullOrEmpty(Account account)
    {
        return account == null || account.Balance == 0;
    }
}
```

And there is the following class, which is analyzed by Flow Analysis's `BD.EXCEPT.NR` rule:

```
public class SomeClass
{
    void SomeMethod()
    {
        Account account = null;
        //some other actions
        if (!AccountManager.IsNullOrEmpty(account))
        {
            Console.WriteLine(account.Balance);
        }
    }
}
```

Flow Analysis assumes that the return value for the method `IsNullOrEmpty()` is unknown. This is because the method is not in the analysis scope. Flow Analysis will analyze the `if` branch and find a violation. Calling the `IsNullOrEmpty()` method, however, returns `true` when the variable is null.

By adding `AccountManager.IsNullOrEmpty()` 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 `if` branch will not be analyzed, resulting in an avoided false positive.

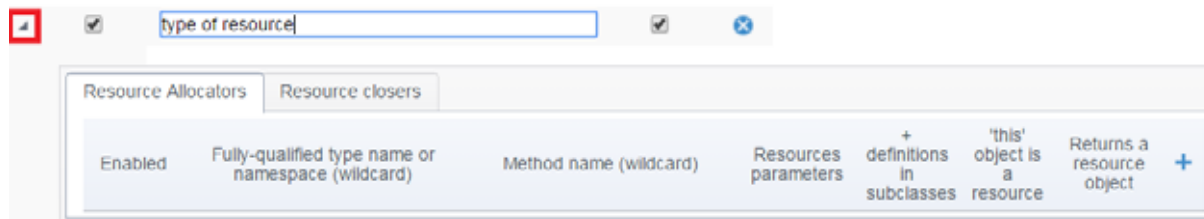
Null-checking Methods Restrictions

Methods added to this parameterization should be static methods that have a primitive `bool` 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. Additionally, null-checking methods can have only one input parameter.

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 or namespace (wildcard):** the fully-qualified name of the type or namespace where the method is declared. Use `*` if you want to describe a function declared in any type or namespace, 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 indicates that 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 or namespace (wildcard):** the fully-qualified name of the type or namespace where the method is declared. Use `*` if you want to describe a function declared in any type or namespace, 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 35, 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.

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 multiple 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
```

Using DTP Engines in an IDE

You can use DTP Engines within Visual Studio. 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 Visual Studio and the DTP Engine Plugin. See the appropriate DTP Plugin User Guide for installation, usage, and other details stored in the [INSTALL]/integration/vstudio/ directory.

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 .NET (see "Parasoft DTP Plugin for [IDE] User's Guide" for additional information). For an overview of the HTML report structure, see "Viewing Reports", page 32.

If the engines are connected to DTP, reports are also sent to the server (see "Sending Results to Development Testing Platform (DTP) Server", page 38).

Specifying Report Output Location

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

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-config "builtin://Demo" -report "C:\Report"
```

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 62, 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 screenshot shows the header of a report. At the top is a blue bar with the text "dotTEST DTP Engine Report" and "DTP Engine for .NET 10.2.3.137" below it. Below this is a section titled "Session Summary" with a table of metadata. To the right of the table is a box containing two rows of summary statistics: "Static Analysis" and "Test Execution" on the left, and "Severity 1 Findings: 9" and "Test Failures: 5/13" on the right.

Session Summary	
Build ID:	dotTEST_10.2.2-2016-05-20
Test Configuration:	user://Run NUnit Tests with coverage
Started:	2016-05-20T14:02:32+02:00
Performed on:	iris by hanna
Session Tag:	\$([control_branch])-win32_x86_64
Project:	dotTEST_10.2.2

Static Analysis	Severity 1 Findings: 9
Test Execution	Test Failures: 5/13

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.

▼ Details - Static Analysis

Static Analysis

Module	Findings			Files		Lines	
	suppressed	total	per 10,000 lines	checked	total	checked	total
com.parasoft.demo	1	804	2213	59	59	3632	3632
Total [0:00:43]	1	804	2213	59	59	3632	3632

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.

All Findings by Category	
Category	Severity
[4] Collections (SO.CO)	
[4] Do not modify collection while iterating over it (SO.CO.ITMOD-1)	
[12] Exceptions (SO.EXCEPT)	
[12] Avoid NullPointerException (SO.EXCEPT.NP-1)	
[3] Optimization (SO.OPT)	
[3] Avoid inefficient removal of Collection elements (SO.OPT.INFCOL-3)	
[1] Avoid inefficient iteration over Map entries (SO.OPT.INFERMAP-3)	
[1] Avoid inefficient removal of Map entries (SO.OPT.INFERMAPB-3)	

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

All Findings by Severity	
Category	Severity
[73] Severity 1 - Highest	
[4] Do not modify collection while iterating over it (SO.CO.ITMOD-1)	
[12] Avoid NullPointerException (SO.EXCEPT.NP-1)	
[2] Avoid use before explicit initialization (SO.PB.NOTEEXPLINT-1)	
[1] Avoid use of fields before initialization in constructors and static initializers (SO.PB.NOTINITCTOR-1)	
[1] Do not append null values to strings (SO.PB.STRNULL-1)	
[1] Avoid division by zero (SO.PB.ZERO-1)	
[1] Protect against Command injection (SO.SECURITY.DCMD-1)	
[1] Protect against Environment injection (SO.SECURITY.DENV-1)	
[1] Protect against File contents injection (SO.SECURITY.DFILES-1)	
[1] Protect against File names injection (SO.SECURITY.DFNAMES-1)	
[1] Protect against Library injection (SO.SECURITY.DLIB-1)	
[1] Protect against Reflection injection (SO.SECURITY.DREF-1)	
[2] Protect against SQL injection (SO.SECURITY.DSQL-1)	
[4] Protect against XML data injection (SO.SECURITY.DXML-1)	
[1] Protect against XSS vulnerabilities (SO.SECURITY.DXSS-1)	
[1] Ensure index is valid in JDBC method invocation (SO.SECURITY.WJBCB-1)	
[1] Use "PreparedStatement" correctly (SO.SECURITY.WJPS-1)	
[6] Unrestricted lock resource (PS.CLOS-1)	
[1] Avoid conditional expressions that always evaluate to a constant value (PS.USC.CC-1)	
[2] Avoid unreachable "else if" and "else" cases (PS.USC.UF-1)	
[18] Do not pass exception messages into output in order to prevent the application from leaking sensitive information (SECURITY.SD.INFO-1)	
[4] Avoid using "SELECT" in SQL queries (SECURITY.WJ.ACCB-1)	
[4] Use "prepareCall" or "prepareStatement" instead of "createStatement" (SECURITY.WJ.ALPS-1)	
[1] Avoid passing hardcoded usernames/passwords/URLs to database connection methods (SECURITY.WJ.HCCB-1)	
[1] Do not define instance fields in Servlet classes (SERVLET.F-1)	
[17] Severity 2 - High	
[6] Avoid conditions that always evaluate to the same value (SO.PB.CC-2)	
[2] Do not use "==" or "!=" to compare objects (PS.OJB.UC-2)	
[1] Avoid using "private" fields which are never given a meaningful value (PS.USC.FCBS-2)	
[4] Avoid "main()" methods because they may allow unauthorized access to classes (SECURITY.WJ.UMAIN-2)	
[4] Use a Context Object to manage HTTP request parameters (SERVLET.UC-2)	
[705] Severity 3 - Medium	
[3] Avoid inefficient removal of Collection elements (SO.OPT.INFCOL-3)	
[1] Avoid inefficient iteration over Map entries (SO.OPT.INFERMAP-3)	

These sections are merged in PDF versions of the report.

- [1] Erratic Application Behavior (SECURITY.EAB)
 - [1] Do not store user-given mutable objects directly into variables (SECURITY.EAB.SMO-1)
- [21] Serialization (SERIAL)
 - [21] Create a "serialVersionUID" for all "Serializable" classes (SERIAL.OUID-3)
- [6] Servlets (SERVLET)
 - [1] Do not define instance fields in Servlet classes (SERVLET.F-1)
 - [4] Use a Context Object to manage HTTP request parameters (SERVLET.UC-2)
 - [1] Do not use JDBC code in Servlet classes (SERVLET.AJDBC-3)
- [25] Unused Code (UC)
 - [10] Avoid unnecessary modifiers in an "interface" (UC.AAI-3)
 - [4] Remove commented out Java code (UC.ACC-3)
 - [11] Avoid local variables that are never read (UC.ALRV-3)

Findings by Category section

- [73] Severity 1 - Highest
 - [4] Do not modify collection while iterating over it (SO.CO.ITMOD-1)
 - [12] Avoid NullPointerException (SO.EXCEPT.NP-1)
 - [2] Avoid use before explicit initialization (SO.PB.NOTEEXPLINT-1)
 - [1] Avoid use of fields before initialization in constructors and static initializers (SO.PB.NOTINITCTOR-1)
 - [1] Do not append null value to strings (SO.PB.STRNULL-1)
 - [1] Avoid division by zero (SO.PB.ZERO-1)
 - [1] Protect against Command injection (SO.SECURITY.DCMD-1)
- [17] Severity 2 - High
 - [6] Avoid conditions that always evaluate to the same value (SO.PB.CC-2)
 - [2] Do not use "==" or "!=" to compare objects (PS.OJB.UC-2)
 - [1] Avoid using "private" fields which are never given a meaningful value (PS.USC.FCBS-2)
 - [4] Avoid "main()" methods because they may allow unauthorized access to classes (SECURITY.WJ.UMAIN-2)
 - [4] Use a Context Object to manage HTTP request parameters (SERVLET.UC-2)

Findings by Severity section

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.

Findings by Author			
Author	Findings		
	Suppressed	Total	Recommended
hanna	0	26	26
hanna Total Findings: 26			
BankExample\Parasoft\DotTest.Examples.Bank.Money\Currency.cs 49: Replace equality comparison of floating-point types to comparison which handles floating-point uncertainty. PB.DNCF-2 BankExample\Parasoft\DotTest.Examples.Bank.Money\CurrencyProvider.cs 23: Provide 'default' for 'switch' statement. CS.PB.DEFSWITCH-2 38: Provide 'default' for 'switch' statement. CS.PB.DEFSWITCH-2 51: Private method 'Parasoft.DotTest.Examples.Bank.Money.CurrencyProvider.Parse(System.String, Parasoft.DotTest.Examples.Bank.Money.CurrencyInfo)' is unused. Call it or remove dead code. CM.UJ.MU.AUPM-2 53: Use TryParse instead of Parse in method 'Parse'. PB.DNUP-2			

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.

Findings by File			
		Expand All	Collapse All
- 26 (0) Total (Suppressed)			
- 26 (0) Parasoft.DotTest.Examples.Bank			
- 3 (0) AccountNumber.cs			
Findings: 3			
14: Change Equals() implementation to match a standard pattern. Use: if (obj == null) { return false; }	hanna	CPU.CPNEG-1	
14: Change Equals() implementation to match a standard pattern. Use: if (obj.GetType() != obj2.GetType()) { return false; }	hanna	CPU.CPNEG-2	
17: Change bitwise operator to logical operator in method 'Equals'.	hanna	CS.PB.BITBOOL-2	

Metrics Summary

If your test configuration includes metrics analysis, a metrics section will appear in the report. See “Metrics Analysis”, page 29, for additional information.

Metrics Summary						
		Expand All	Collapse All	Back to Top		
Metric name	Number of Items	Average	Std. Deviation	Maximum	Minimum	
- McCabe Cyclomatic Complexity (METRIC.CC)	59	1,542	0,908	4	1	
Parasoft.DotTest.Examples.Bank	59	1,542	0,908	4	1	
- Nested Blocks Depth (METRIC.NBD)	113	0,248	0,588	3	0	
Parasoft.DotTest.Examples.Bank	113	0,248	0,588	3	0	
- Number of Physical Lines in Files (METRIC.NOPLIF)	19	55,526	27,886	141	13	
Parasoft.DotTest.Examples.Bank	19	55,526	27,886	141	13	
- Number of Source Lines in Methods (METRIC.NOSLIM)	87	6,885	6,439	48	1	
Parasoft.DotTest.Examples.Bank	87	6,885	6,439	48	1	

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

(5) Unit Test Problems
(5) Assertion Failures
(5) Error

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
Yatna	5	5
<hr/>		
hanna Total Findings : 5 Back to Top		
<pre>BankExample.Parasoft.DotTest.Examples.Bank.BankUser.cs [5] Test case: Parasoft.DotTest.Examples.Bank.Tests.BankUserTests.Test"foo bar" System.ArgumentException : parsing "'[a-z]+[0-9]+'(?!'[a-z]+[0-9]+' '@'[a-z]+[0-9]+' '@A-Z[0-9]')'- Unrecognized escape sequence '_'. (Source: System.Text.RegularExpressions.Regex.Parser.ScanChar(Escape) System.Text.RegularExpressions.Regex.Parser.ScanCharClass(Boolean caseInsensitive, Boolean scanOnly) System.Text.RegularExpressions.Regex.Parser.CountCaptures() System.Text.RegularExpressions.Regex.Parser.Parse(String re, RegexOptions options) System.Text.RegularExpressions.Regex.Compile(String pattern, RegexOptions options, TimeSpan matchTimeout, Boolean useCache) System.Text.RegularExpressions.Regex.IsMatch(String input, String pattern, RegexOptions options, TimeSpan matchTimeout) System.Text.RegularExpressions.Regex.IsMatch(String input, String pattern) Parasoft.DotTest.Examples.Bank.BankUser.CheckEmail() Parasoft.DotTest.Examples.Bank.Tests.BankUserTests.Test(String sourceMail)</pre>		

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.

Executed Tests (Details) [Expand All](#) [Collapse All](#) [Back to Top](#)

<ul style="list-style-type: none"> - [8/13] [0:00:00.214] Passed : Total - [8/13] [0:00:00.214] Parasoft.DotTest.Examples.Bank.Tests <ul style="list-style-type: none"> - [P] [0:00:00.104] Parasoft.DotTest.Examples.Bank.Tests.AccountNumberTests.TestAccountNumberCreation - [P] [0:00:00.010] Parasoft.DotTest.Examples.Bank.Tests.AccountNumberTests.TestAccountNumberHashing - [P] [0:00:00.010] Parasoft.DotTest.Examples.Bank.Tests.AccountNumberTests.TestInvalidAccountNumberCreation - [F] [0:00:00.050] Parasoft.DotTest.Examples.Bank.Tests.BankUserTests.Test - [P] [0:00:00.034] Parasoft.DotTest.Examples.Bank.Tests.CurrencyExchangeConverterTests

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.

Coverage [Expand All](#) [Collapse All](#) [Back to Top](#)

<ul style="list-style-type: none"> - Total [22% 109/496 executable lines] - Parasoft.DotTest.Examples.Bank [22% 109/496 executable lines] - Money [22% 28/125 executable lines] - Currency.cs [19% 5/26 executable lines] - CurrencyInfo.cs [52% 8/13 executable lines] <ul style="list-style-type: none"> CurrencyInfo.CurrencyInfo(System.String, System.String, System.String) [70% 7/10 executable lines] CurrencyInfo.get_Code() [100% 1/1 executable lines] CurrencyInfo.get_Notation() [0% 0/1 executable lines] CurrencyInfo.get_Sign() [0% 0/1 executable lines]

Test Parameters

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

Test Parameters

```
dottestcli -solution C:\dotTEST\Examples_10.2.3.137\VS2013\030_NUnit\BankExample\BankExample.sln -config builtin:Run NUnit Tests with coverage -report C:\dotTEST\Examples_10.2.3.137\VS2013\030_NUnit\020_Run_NUnit_Tests_With_Coverage_Report -out C:\dotTEST\Examples_10.2.3.137\VS2013\030_NUnit\020_Run_NUnit_Tests_With_Coverage_Output.txt
```

Sending Results to Development Testing Platform (DTP) Server

See “Connecting to DTP Server”, page 10, for information about configuring your connection to DTP Server. Use the `-publish` switch to report test results to DTP server.

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"
-config "builtin://Demo" -publish
```

Associating Results with a DTP Project

Configure the `dtp.project` property to associate results with a project in Development Testing Platform.

```
dtp.project=[PROJECT NAME]
```

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 54, 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 15).
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 .NET currently ships with out-of-the-box support for the following unit testing tools:

- NUnit
- MS Test versions VS2010, VS2012, VS2013, VS2015

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

Running Unit Tests with Coverage

You can run NUnit tests with or without coverage analysis. Coverage information is collected by running a dedicated test configuration. The built-in test configuration for collecting coverage is `Run NUnit Tests with coverage`.

Running NUnit Tests

Run one of the built-in test configurations to run NUnit tests on the scope of analysis. See “Specifying Test Configurations”, page 15 for details on using the `-config` switch.

Examples

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"  
-config "builtin://Run NUnit Tests" -report "C:\Report"
```

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"  
-config "builtin://Run NUnit Tests with coverage" -report "C:\Report"
```

All files associated with NUnit integration are in the `[INSTALL_DIR]\integration\NUnit-2.6.3` directory.

Running MSTest Tests

1. MSTest must be integrated with UTC to run MSTest tests. If you did not integrate with MSTest during installation, you can run `Deploy.exe` in the MSTest integration directory:
`[INSTALLATION_DIR]\integration\MSTest\Deploy.exe`
2. Run one of the built-in test configurations for MSTest test and extend your `dottestcli.exe` command to include the build script that invokes `mstest.exe`:

```
dottestcli.exe -solution "FooSolution.sln" -config "builtin://Execute MSTests with  
Coverage" -report "Report" -- "Run_tests.bat"
```

The `--` separator indicates the end of the dotTEST command line; arguments following `--` form a command line that is invoked by `dottestcli.exe`.

In the above example, dotTEST launches the `Run_tests.bat` script and listens to all MSTests executed in the script as it runs. Unit test results are collected and code coverage is measured for `FooSolution.sln`.

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 `NUnit.Framework.PropertyAttribute` attribute.

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 62 for additional information.

Using NUnit Attributes

Use the `NUnit.Framework.PropertyAttribute` attribute to associate NUnit tests with artifacts. See the NUnit documentation for additional information about the `PropertyAttribute` attribute: <http://www.nunit.org/index.php?p=property&r=2.6.3>.

Place the tag in the `NUnit.Framework.PropertyAttribute` attribute to associate it with your tests.

```
[Property("bug", "1234")]
[Test]
public void Test()
{
    ...
}
```

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, bug 9876 is associated with both tests within the `Test` class, whereas bug 111 is associated only with the `Test2` test.

```
[Property("bug", "9876")]
public class Tests
{
    [Test]
    public void Test1()
    {
        ...
    }

    [Property("bug", "111")]
    [Test]
    public void Test2()
    {
        ...
    }
}
```

Multiple Associations

You can associate one tag with more than one artifact.

```
[Property("bug", "1234, 1199")]  
[Test]  
public void Test()  
{  
    ...  
}
```

You can separate the tasks with a comma, a semicolon or a space character. In the example below, the test is associated with all the listed tasks:

```
[Property("bug", "1234, 1199; 2345 1928")]  
[Test]  
public void Test()  
{  
    ...  
}
```

Code Coverage Engine

In this section:

- Application Coverage for Standalone Applications
- Application Coverage for Web Applications

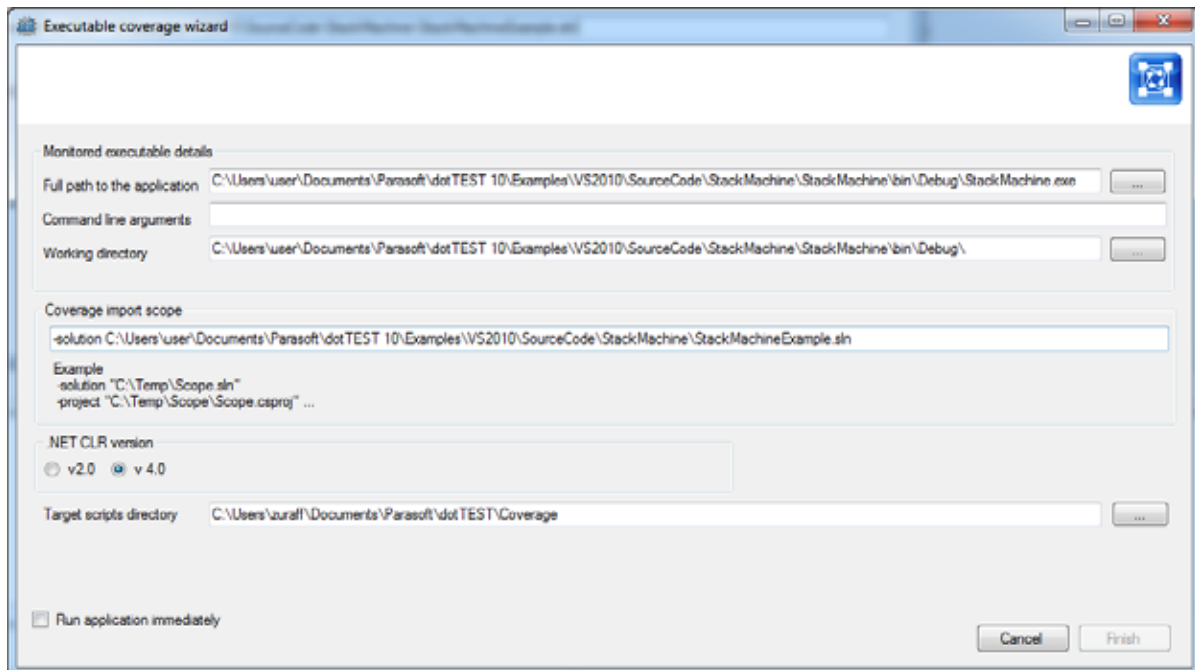
Application Coverage for Standalone Applications

DTP Engines for .NET ship with the `coverage.exe` tool that facilitates collecting coverage information during execution of standalone applications. The tool creates two *.bat scripts: one script runs and monitors the application and the other script imports and reports coverage data.

Running `coverage.exe` launches a wizard that allows you to specify the necessary information in the GUI. Alternatively, you can provide the information with the dedicated command line options.

Using the Coverage Wizard

1. Run the `[INSTALL_DIR]\coverage.exe` tool and specify the following information:
 - Full path to the application
 - Command line arguments
 - Working directory



2. Define scope of coverage to import by providing `-solution` or `-project` switches that will be passed to DTP Engine executable so that it can locate sources. See “Defining Test Scope”, page 18, for more information about the switches.
3. Specify the .NET CLR version used by the application.
4. Specify a directory for the scripts generated by the wizard.
5. Enable the **Run application immediately** option to automatically launch the `monitorCoverage.bat` script (optional; you can also open the directory specified in step 4 and manually run the script).
6. Execute your test cases and close the application.
7. Run the `importCoverage.bat` script (in the directory you specified in the wizard).

8. Check the report for coverage information. The report is written to the 'report' folder by default.

Important

By default, the `monitorCoverage.bat` script cleans the contents of the directory that contains the logs of the previous execution. If you do not want the logs to be removed, you can modify the `monitorCoverage.bat` script by removing the `rmdir /s /q [path/to/directory]` command.

Alternatively, you can comment the command with the `rem` tag, for example:

```
rem rmdir /s /q "%userprofile%\Documents\Parasoft\dotTEST\Coverage\CoverageLogs"
```

Using the Command Line Options

You can run `coverage.exe` with the dedicated command line options to specify the information necessary for collecting application coverage :

```
coverage.exe -app [path] -workingDir [path] -appArgs [arguments] -commandsDir [path] -scope [dotTEST scope switch] -clr [version] -run
```

The following options are available:

Option Name	Value	Description
-app	[path]	The full path to the application
-workingDir	[path]	The path to the working directory
-appArgs	[argument]	The application command line arguments
-commandsDir	[path]	Specifies the directory for the *.bat scripts generated by the coverage tool
-scope	-solution [path] -project [path]	Specifies the scope of the coverage to import by providing -solution or -project switches that will be passed to DTP Engine executable so that it can locate sources. See “Defining Test Scope”, page 18, for more information about the switches. Example: -scope "-solution C:\temp\Scope.sln" -scope "-project C:\temp\Scope.csproj"
-clr	v20 v40	Specifies the .NET CLR version used by the application
-run	none	Automatically launches the <code>monitorCoverage.bat</code> script (optional; you can also open the directory specified with the <code>-commandDir</code> option and manually run the script).

You can also collect coverage information during test execution of running web applications. See “Application Coverage for Web Applications”, page 47 for details.

For information about collecting coverage for unit tests, see “Unit Test Connector”, page 39.

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 for Web Applications

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:

- Internet Information Services (IIS) version 7.5 or higher
- Coverage Agent Manager (CAM) (contact your Parasoft representative) or SOAtest

Process Overview

The DTP Engine for .NET 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 47.
2. The coverage agent must be attached to the AUT. See "Attaching the Coverage Agent to the AUT", page 48.

Generating the Static Coverage File

If you use a source control system, ensure that your source control settings are properly configured; see "Settings Reference", page 54.

Run the following test configuration on the solution:

```
dottestcli.exe -config "builtin://Collect Static Coverage" -solution SOLUTION_PATH
```

The `dottestcli` console output will indicate where the static coverage data is saved:

```
Saving static coverage information into:
'C:\Users\[USER]\Documents\Parasoft\dotTEST\Coverage\Static\[FILE].data'
```

Customizing Scope of Coverage

By default, coverage is measured for the entire web application. You can customize the scope of coverage by adding the following switches when collecting static coverage to measure specific parts of the application (see “Defining Test Scope”, page 18, for usage information):

```
dottestcli.exe -config "builtin://Collect Static Coverage"
-solution "C:\Devel\FooSolution\FooSolution.sln"
-resource "FooSolution/QuxProject"
-include "C:\Devel\FooSolution\src\QuxProject\**\*.cs"
-exclude "C:\Devel\FooSolution\src\QuxProject\**\tests\**\*.cs"
```

The `-resource` switch points to a path inside the solution, while the `-include` and `-exclude` switches should be paths in the file system.

The scope information is stored in a scope configuration file, which can be provided to the IIS manager tool during web server configuration (see “Attaching the Coverage Agent to the AUT”, page 48). The output from the console will indicate the location of the scope configuration file:

```
Saving static coverage scope configuration into: 'C:\Users\[USER]\Documents\Parasoft\dot-
TEST\Coverage\Static\[FILE].txt'
```

It is not possible to use the application coverage scope file for web projects that are compiled on IIS. This is because the target assemblies of IIS compilations are named unpredictably. Scope files can be used safely when the assembly name loaded by IIS can be predetermined before coverage collection starts.

Attaching the Coverage Agent to the AUT

1. Copy the `[INSTALLATION_DIR]\integration\IIS` directory to the machine where IIS is installed and the web application is deployed.
2. Run a console as an Administrator
3. Invoke the `dotTEST` IIS Manager tool on this machine to enable runtime coverage collection inside IIS:

```
dottest_iismanager.exe
```

If you’ve specified a coverage scope, you must also include the path to the scope configuration file:

```
dottest_iismanager.exe -scope [FILE_PATH]
```

You can add the `-agentTimeout` switch to change the default timeout (1500 ms) for connection with the Coverage Agent. Adjust the timeout to your machine capabilities and provide the value in milliseconds:

```
dottest_iismanager.exe -agentTimeout [TIMEOUT_IN_MILLISECONDS]
```

If you provide 0 or a negative value, the connection attempt will not timeout, which may lead to a considerable slowdown or hang the tool.

You can also add the `-port` switch to change the port number when you start `dottest_iismanager` if the default port is unavailable:

```
dottest_iismanager.exe -port [PORT]
```

`dottest_iismanager` initializes the environment for the web server (IIS) and behaves like a service, enabling you to execute tests and collect coverage. The service is ready and waiting for commands as long as the following message is printed to the output:

```
Write 'exit' and hit Enter to close dottest_iismanager
```

Be aware that a test session and test can be started even if the tested website or application has not been loaded yet.

4. Ensure that port 8050 (default port for the coverage agent) allows HTTP traffic in firewall settings on this machine. You can change the coverage agent port number if the default port is unavailable.
5. Open the web site or application
6. Go to the following address to check the status of the coverage agent:

```
http://host:8050/status
```

You should receive the following response:

```
{"session":null,"test":null}
```

Collecting Coverage from Multiple Users

You can collect coverage information for multiple users that are simultaneously accessing the same web application server. This requires launching the dotTEST IIS Manager the `-multiuser` switch:

```
dottest_iismanager.exe -multiuser
```

See the Coverage Agent Manager (CAM) section of the DTP documentation for details.

Changing IIS Idle Time-outs

By default, IIS application pool processes are recycled after 20 minutes of idle time, which can have negative consequences on a test session. You can prevent this behavior by changing the default value so that people working with the application do not experience unexpected stops and restarts during a test session.

1. Start the Internet Information Services (IIS) Manager
2. Open the Application Pools node
3. Choose the pool for your web application
4. Click **Advanced Settings** in the Actions panel

5. In Process Model section, change the Idle Time-out (minutes) setting to a value better-suited to your testing practices.

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 specified 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 `dottestcli.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 10
- "Sending Results to Development Testing Platform (DTP) Server", page 38
- "Settings Reference", page 54

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. Copy the runtime coverage and static coverage files to a directory on the same machine and run `dottestcli` using the `-runtimeCoverage` and `-staticCoverage` switches to specify the location of the files:

```
dottestcli.exe -runtimeCoverage [path] -report [path] -publish -settings [path]
-out [path] -staticCoverage [path]
```

Stopping Dynamic Coverage Data Collection

In the open console, type `exit` while the following message is printed to the output to stop `dottest_iismanager`:

Write 'exit' and hit Enter to close `dottest_iismanager`

You can also send a request to the service by entering the following URL in the browser:

`http://host:port/shutdown`

Stop `dottest_iismanager` only when all test sessions are finished. Application coverage will no longer be collected when the service stops, so it is important that `dottest_iismanager` runs continuously while performing tests to collect coverage.

If any errors occur when `dottest_iismanager` exits that prevent the clean-up of the Web Server environment, then execute `dottest_iismanager` and include the `-stop` parameter to bring back the original Web Server environment and settings:

```
dottest_iismanager.exe -stop
```

Known Limitations

- You can download coverage information that was collected in a test session. Coverage data collected when no test session was active cannot be downloaded.
- 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 multiuser mode must be enabled (see “Collecting Coverage from Multiple Users”, page 49).
- 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, collecting coverage for WCF-based applications requires that they have the ASP.NET compatibility mode enabled.
- 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 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.

Customizing DTP Engines for .NET

Basic settings can be specified in the `dottestcli.properties` file in the installation directory. By default, most settings are disabled. Uncomment the settings you want to enable and specify the parameters. Modifying the properties file may require administrator access.

You can create custom properties files and point to them with the `-settings` switch.

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"  
-settings "C:\Devel\Settings\ntp_server.properties"
```

The `-settings` switch may be specified multiple times. Entries with the same key will be overwritten.

```
dottestcli.exe -solution "C:\Devel\FooSolution\FooSolution.sln"  
-settings "C:\Devel\Settings\ntp_server.properties"  
-settings "C:\Devel\Settings\email_server.properties"
```

General settings are applied in the following order:

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

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

If you have dotTest 9.5 or later, settings can be prepared in Visual Studio and exported to a `*.properties` file.

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

Auto-configuring Settings from DTP Server

You can specify settings in Development Testing Platform and configure the Static Analysis Engine to use the settings when it connects to DTP Server. This enables you to use the same settings across multiple machines and configurations to ensure consistency. Edit the following setting in the `dottestcli.properties` file to enable auto-configuration:

```
ntp.autoconfig=true
```

Settings can be auto-configured for the entire organization or per project.

Specifying Organization-wide Settings

1. Log into Development Testing Platform with administrator credentials

2. Switch to Report Center and click the **administration** link.



3. Choose **Settings> Parasoft Test** and enter the settings from `dottestcli.properties`
4. Click **Save**.

Specifying Settings Per Project

1. Log into Development Testing Platform with administrator credentials
2. Switch to Report Center and click the **administration** link.
3. Choose **Projects> Search** and click **Search**
4. Click on your project name in the results and click the **Parasoft Test Settings** tab
5. Enter the settings from `dottestcli.properties` and click **Save**.

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
<code>analysis_type</code>	Outputs a comma separated list of enabled analysis types (e.g., Static, Generation and Execution)	<code>\${analysis_type}</code>
<code>env_var</code>	Outputs the value of the environmental variable specified after the colon.	<code>\${env_var:HOME}</code>
<code>config_name</code>	Outputs the name of executed Test Configuration.	<code>\${config_name}</code>
<code>dtp_project</code>	Outputs the name of DTP project specified in the settings file using <code>dtp.project</code> option.	<code>\${dtp_project}</code>
<code>project_module</code>	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 <code>project.module</code> option.	<code>\${module_name}</code>
<code>host_name</code>	Outputs the name of the host.	<code>\${host_name}</code>
<code>user_name</code>	Outputs the name of the current user.	<code>\${user_name}</code>
<code>os</code>	Outputs the name of the operating system.	<code>\${os}</code>

Variable	Description	Example
arch	Outputs the name of the operating system architecture	<code>\${arch}</code>
exec_env	Outputs the execution environment. This variable is a concatenation of <code>\${os}</code> and <code>\${arch}</code> variables. It can be configured in the settings file with the <code>exec.env</code> option.	<code>\${exec_env}</code>
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 <code>scontrol.branch</code> option.	<code>\${scontrol_branch}</code>
tool_name	Outputs the name of the tool (i.e., Jtest, C++test, dotTEST).	<code>\${tool_name}</code>
jvm_prop	Outputs the value of the Java vm property specified after the colon.	<code>\${jvm_prop:os.name}</code>

Settings Reference

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

Base Configuration Settings

Setting	Value	Description/Notes
<code>console.verbosity.level</code>	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.
<code>parallel.mode</code>	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.

Setting	Value	Description/Notes
<code>parallel.no_memory_limit</code>	true false	Enables/disables restrictions (beyond existing system limitations) on the memory consumed by parallel processing. Default is <code>false</code>
<code>parallel.free_memory_limit</code>	[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
<code>parallel.max_threads</code>	[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
<code>file.encoding.mode</code>	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 <code>file.encoding.lang</code> user: enables use of specified encoding by <code>file.encoding.name</code> .
<code>file.encoding.lang</code>	[code]	Allows specify language's numeric code when <code>file.encoding.mode</code> is set to <code>auto</code> : Japanese = 1 Chinese = 2 Simplified Chinese = 3 Traditional Chinese = 4 Korean = 5

Setting	Value	Description/Notes
<code>file.encoding.name</code>	[encoding]	Allows you to specify the encoding name when <code>file.encoding.mode</code> is set to user: ASCII-US UTF-8 UTF-16 UTF-16LE UTF-16BE ...
<code>settings.validation</code>	true false	Enables/disables settings validation.
<code>settings.rules.file.dottest</code>	path	Indicates the path to a file that contains additional rules for settings validation. The file should follow the <code>.properties</code> format and include rules according to the following examples: <pre>engine.path=\$ANY engine.enabled=\$BOOLEAN engine.analysis.deep=\$INTEGER engine.severity.limit=\$REGEXP{ [1-5] } engine.verbosity.level=\$REGEXP_IC{ (low) (normal) (high) }</pre>
<code>dottest.build.nobuild</code>	true false	Disables/ enables build of the tested solution or projects. If enabled, the building phase is skipped during analysis. The default is <code>false</code> .

Test Configuration Settings

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

Development Testing Platform Settings

Setting	Value	Description/Notes
<code>dtp.server</code>	<code>[host]</code>	Specifies the host name of the DTP server.
<code>dtp.port</code>	<code>[port]</code>	Specifies the port number on DTP server port. The default settings is 443.
<code>dtp.user</code> <code>dtp.password</code>	<code>[username]</code> <code>[password]</code>	Specifies authentication to connect to DTP server.
<code>dtp.project</code>	<code>[project_name]</code>	Specifies the name of the DTP project that you want linked to. This settings is optional.
<code>dtp.autoconfig</code>	<code>true</code> <code>false</code>	Enables auto configuration with settings stored on the DTP server. The default is <code>false</code> .
<code>report.dtp.publish</code>	<code>true</code> <code>false</code>	Determines whether the current installation is reporting test results to DTP server. The default is <code>false</code> .
<code>report.dtp.publish.src</code>	<code>off</code> <code>min</code> <code>full</code>	Determines whether tested source code is published to DTP server. <code>off</code> : code is not published to DTP server. <code>min</code> : publishes minimal part of sources. In most cases, source code without references to source control, e.g., auto-generated code, is published. <code>full</code> : publishes all sources associated with the specified scope. The default is <code>full</code> if <code>report.dtp.publish</code> is enabled, otherwise the default is <code>off</code>
<code>dtp.share.enabled</code>	<code>true</code> <code>false</code>	Enables/disables connection to Team Server. The default is <code>false</code> .
<code>dottest.license.use_network</code>	<code>true</code> <code>false</code>	Enables/disables license retrieval from License Service. The default setting is <code>true</code> .

Setting	Value	Description/Notes
<code>dottest.license.network.type</code>	<code>ntp</code> <code>ls</code>	Sets the network license type. <code>ntp</code> : file count license that limits usage to a certain number of files as determined by your licensing agreement <code>ls</code> : floating license (machine locked) that limits usage to a certain number of machines
<code>dottest.license.local.password</code>	<code>[password]</code>	Specifies the local license password.
<code>dottest.license.local.expiration</code>	<code>[expiration]</code>	Specifies the local license expiration date.
<code>dottest.license.network.edition</code>	<code>desktop_edition</code> <code>server_edition</code> <code>custom_edition</code>	Specifies the type of license that will be retrieved from License Service for this installation. Default is <code>custom_edition</code>
<code>dottest.license.custom_edition_features</code>	<code>[feature_name, ...]</code>	Specifies active features for custom license edition.
<code>dottest.license.wait.for.tokens.time</code>	<code>[minutes]</code>	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
<code>scope.local</code>	<code>true</code> <code>false</code>	Enables/disables code authorship computation based on the local user and system files modification time. Default is <code>true</code>
<code>scope.scontrol</code>	<code>true</code> <code>false</code>	Enables/disables code authorship computation based on data from a supported source control system. Default is <code>false</code>
<code>scope.xmlmap</code>	<code>true</code> <code>false</code>	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 21 for syntax information. Default is <code>false</code>
<code>scope.xmlmap.file</code>	<code>[path]</code>	Specifies the path to XML mapping file that defines how tasks should be assigned for particular files or sets of files.

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

Suppression Settings

Setting	Value	Description/Notes
<code>suppression{n}.file.ext</code>	[ext]	Specifies the extension of types of files that should be scanned for comment suppressions. Example: <code>suppression1.file.ext=xml</code> <code>suppression2.file.ext=java</code> Set the comment prefix with the <code>suppression{n}.comment</code> setting.
<code>suppression{n}.comment</code>	[comment]	Specifies comment prefix for types of files identified in <code>suppression.file.ext</code> setting. Example: <code>suppression1.comment=//</code> <code>suppression2.comment=<!--</code>
<code>suppression{n}.comment.suffix</code>	[comment suffix]	Defines the suppression comment suffix when file extensions has been specified with the <code>suppression.file.ext</code> setting. If not specified then suppression comments will not be suffixed. Example: <code>suppression1.comment.suffix=--></code>
<code>suppression{n}.block.only</code>	true fales	Enables/disables block-only comment suppressions support when file extensions have been specified with the <code>suppression.file.ext</code> setting. Default is false.
<code>suppression.local.dir</code>	[path]	Specifies the custom directory for storing local suppressions. An absolute path should be provided. Example: <code>suppression.local.dir=file:///C:/parasoft/suppression/storage</code> <code>suppression.local.dir=../suppression/storage</code>

Technical Support Settings

Setting	Value	Description/Notes
<code>techsupport.enabled</code>	true false	Enables/disables global automatic technical support data collection is globally enabled with verbose logging. Default is false
<code>logging.verbose</code>	true false	Enables/disables verbose logs. Verbose logs are stored in the <code>xtest.log</code> file in the location specified with the <code>local.storage.dir</code> 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
<code>logging.scontrol.verbose</code>	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
<code>techsupport.create.on.exit</code>	true false	Enables/disables automatic archive creation when the application is shut down. The <code>techsupport.enabled</code> setting must also be enabled for packages to be created automatically. Default is true.
<code>techsupport.archive.location</code>	[path]	Specifies the custom directory where support packages should be created.
<code>techsupport.include.reports</code>	true false	Enables/disables the inclusion of reports in the technical support package.

Report Settings

Setting	Value	Description/Notes
<code>session.tag</code>	[name]	Specifies a tag for signing results from 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.
<code>build.id</code>	[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 <code>build-YYYY-MM-dd HH:mm:ss</code>
<code>project.module</code>	[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.
<code>exec.env</code>	[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 <code>exec.env</code> 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.
<code>report.location</code>	[path]	Specifies the directory where report should be created.
<code>report.format</code>	xml html pdf csv custom	Specifies the report format. Use a comma separated list of formats to generate multiple formats. Default is <code>xml</code>

Setting	Value	Description/Notes
<code>report.custom.extension</code>	[ext]	Specifies the report file extension of the XSL file for a custom format. Use with <code>report.format=custom</code> and <code>report.custom.xsl.file</code> .
<code>report.custom.xsl.file</code>	[path]	Specifies the location of the XSL file for a custom format. Use with <code>report.format=custom</code> and <code>report.custom.extension</code>
<code>report.developer_errors</code>	true false	Determines whether manager reports include details about developer errors. The default is true.
<code>report.developer_reports</code>	true false	Determines whether the system generates detailed reports for all developers (in addition to a summary report for managers). The default is false.
<code>report.authors_details</code>	true false	Determines whether the report includes an overview of the number and type of tasks assigned to each developer. The default is true.
<code>report.contexts_details</code>	true false	Determines whether the report includes an overview of the files that were checked or executed during testing. The default is true.
<code>report.suppressed_msgs</code>	true false	Determines whether report includes suppressed messages. The default setting is false.
<code>report.metadata</code>	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 setting to have an effect. Default is true.

Setting	Value	Description/Notes
<code>report.scontrol</code>	<code>off</code> <code>min</code> <code>full</code>	Specifies if and how much additional information from source control is included in the report: <code>min</code> : repositories, file paths and revisions <code>full</code> : includes the same information as <code>min</code> , as well as task revisions and comments. Default is <code>off</code>
<code>report.associations</code>	<code>true</code> <code>false</code>	Enables/disables showing requirements, defects, tasks, and feature requests associated with a test in the report. The default is <code>true</code> .
<code>issue.tracking.tags</code>	<code>[tag1,tag2,...]</code>	Specifies a list of issue tracking tags. The following tags are supported by default: <code>pr</code> , <code>fr</code> , <code>task</code> , <code>asset</code> , <code>req</code> .
<code>report.assoc.url.[tag]</code>	<code>[url]</code>	Generates link to association inside the HTML report. The URL is a query string containing an <code>[%ID%]</code> placeholder for the <code>PropertyAttribute</code> value.
<code>report.active_rules</code>	<code>true</code> <code>false</code>	Determines if report contains a list of the rules that were enabled for the test. The default setting is <code>true</code> .
<code>report.rules</code>	<code>[url]</code>	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: <code>report.rules=file:///C:/parasoft/gendoc/</code> <code>report.rules=./gendoc/</code>
<code>report.test_params</code>	<code>true</code> <code>false</code>	Determines whether report includes test parameter details. The default setting is <code>true</code> .
<code>report.coverage.images</code>	<code>[tag1, . . .]</code>	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.

Setting	Value	Description/Notes
<code>report.coverage.limit</code>	[limit]	Value that specifies the lower coverage threshold. Coverage results lower than this value are highlighted in the report. Default is 40
<code>report.metrics.attributes</code>	[attr1;attr2;...]	Specifies a list of additional attributes for metric results. The following attributes are supported by default: <code>module</code> , <code>namespace</code> , <code>type</code> , <code>method</code> .
<code>report.archive</code>	true false	Enables/disables archiving reports into a ZIP file.
<code>report.graph.start_date</code>	[MM/dd/yy]	Specifies start date for trend graphs that track static analysis task, test execution, and coverage. Use with <code>report.graph.period=[?d ?m ?y]</code>
<code>report.graph.period</code>	[?d ?m ?y]	Determines the duration from the start date for trend graphs that track static analysis task, test execution, and coverage. Use with <code>report.graph.start_date=[MM/dd/yy]</code>
<code>report.mail.enabled</code>	true false	Enables/disables report emails to developers and additional recipients specified with the <code>report.mail.cc</code> 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 <code>false</code> .
<code>report.mail.server</code>	[host]	Specifies the mail server used to send reports.
<code>report.mail.port</code>	[port]	Specifies the port for SMTP server. The default port is 25.
<code>report.mail.security</code>	[security]	Specifies SMTP server connection security. STARTTLS and SSL are supported. The default is STARTTLS.
<code>report.mail.subject</code>	[subject line]	Specifies the subject line of the emails sent.

Setting	Value	Description/Notes
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.
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.

Setting	Value	Description/Notes
<code>report.mail.include</code>	[email, ...]	<p>Specifies email addresses of developers that you want to receive developer reports. Multiple addresses must be separated with a semicolon.</p> <p>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).</p> <p>This setting overrides addresses specified in the 'exclude' list.</p>
<code>report.mail.exclude</code>	[email; ...]	Specifies email addresses that should be excluded from automatically receiving reports.
<code>report.mail.exclude.developers</code>	true false	<p>Enables/disables report emails to developers not explicitly listed in the <code>report.mail.cc</code> setting. This setting is used to prevent reports from being mailed to individual developers.</p> <p>The default setting is <code>false</code>.</p>
<code>report.mail.unknown</code>	[email user_name]	Specifies where to email reports for errors assigned to "unknown".
<code>report.mail.on.error.only</code>	true false	<p>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.</p> <p>The default setting is <code>false</code>.</p>
<code>report.setup.problems</code>	top bottom hidden	<p>Determines placement of setup problems section in report.</p> <p>The default setting is <code>bottom</code>.</p>
<code>report.setup.problems.category_limit</code>	[numerical value]	<p>Specifies a limit to the number of messages reported in a single setup problem category.</p> <p>Default is 10</p>
<code>report.setup.problems.display_limit</code>	[numerical value]	<p>Specifies a limit to the total number of messages displayed in the HTML report in the setup problem section.</p> <p>Default is 100</p>

Setting	Value	Description/Notes
<code>report.setup.problems.console</code>	<code>true</code> <code>false</code>	Determines whether setup problems will be printed on the console. The default setting is <code>true</code> .
<code>report.ue_coverage_details.htmls</code>	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
<code>report.separate_vm.xmx</code>	[size]	Specifies how much memory should be used for reports generation. The default is 1024M.
<code>report.separate_vm</code>	<code>true</code> <code>false</code>	Enables/disables generating reports as a separate virtual machine. Default is <code>false</code> .
<code>report.separate_vm.launch.file</code>	[path]	Specifies path to launch file which should be used during reports generation.
<code>dupcode.sorting.mode</code>	<code>oldest</code> <code>newest</code> <code>paths</code>	Determines how elements in the code duplication findings are sorted. <code>oldest</code> : the oldest result appears at the top. <code>newest</code> : the newest result appears at the top. <code>paths</code> : sorts by full path names in ascending alphabetical order (A to Z). The default is <code>paths</code> .
<code>report.coverage.version</code>	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
<code>scontrol.timeout</code>	[seconds]	Specifies timeout value for operations with source control. The default value is 60.
<code>scontrol.branch</code>	[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
<code>scontrol.rep{n}.type</code>	accurev	AccuRev repository type identifier.
<code>scontrol.accurev.exec</code>	[path]	Path to external client executable (accurev).
<code>scontrol.rep{n}.accurev.host</code>	[host]	AccuRev server host.
<code>scontrol.rep{n}.accurev.port</code>	[port]	AccuRev server port. Default port is 1666.
<code>scontrol.rep{n}.accurev.login</code>	[login]	AccuRev user name.
<code>scontrol.rep{n}.accurev.password</code>	[password]	AccuRev password.

ClearCase Source Control Settings

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

CVS Source Control Settings

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

Setting Name	Value	Description/Notes
<code>scontrol.rep{n}.cvs.ssh.passphrase</code>	[passphrase]	Specifies the passphrase for SSH connections with the key authentication mechanism.
<code>scontrol.rep{n}.cvs.useShell</code>	true false	Enables/disables an external program (CVS_RSH) to establish a connection to the CVS repository. Default is <code>false</code> .
<code>scontrol.rep{n}.cvs.ext.shell</code>	[path]	Specifies the path to the executable to be used as the CVS_RSH program. Command line parameters should be specified in the <code>cvs.ext.params</code> property.
<code>scontrol.rep{n}.cvs.ext.params</code>	[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 <code>cvs.ssh.loginname</code> {extpassword} parameter <code>cvs.ssh.password</code> {keyfile} parameter <code>cvs.ssh.keyfile</code> {passphrase} parameter <code>cvs.ssh.passphrase</code>

Git Source Control Settings

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

Setting Name	Value	Description/Notes
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 repository type identifier.
scontrol.hg.exec	[path]	Path to external client executable. Default 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.

Setting Name	Value	Description/Notes
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.
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).

Setting Name	Value	Description/Notes
<code>scontrol.rep{n}.starteam.path</code>	[path]	<p>Specifies the project, view, or folder that you are currently working with.</p> <p>You can specify a project name (all views will be scanned when searching for the repository path), project/view (only the given view will be 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.</p> <p>Examples:</p> <pre>scontrol.rep.starteam.path=proj1</pre> <pre>scontrol.rep.starteam.path=proj1/view1</pre> <pre>scontrol.rep.starteam.path=proj1/view1/folderA</pre> <pre>scontrol.rep.starteam.path=proj1/view1/folderA/folderB</pre>
<code>scontrol.rep{n}.starteam.workdir</code>	[path]	<p>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 <code>scontrol.rep.starteam.path</code> setting points to a StarTeam view or folder.</p> <p>Examples:</p> <pre>scontrol.rep.starteam.workdir=C:\\storage\\dv</pre> <pre>scontrol.rep.starteam.workdir=/home/storage/dv</pre>

Subversion Source Control Settings

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

Synergy/CM Source Control Settings

Setting Name	Value	Description/Notes
<code>scontrol.rep{n}.type</code>	<code>synergy</code>	Synergy/CM repository type identifier.
<code>scontrol.synergy.exec</code>	<code>[path]</code>	Path to external client executable (ccm).
<code>scontrol.rep{n}.synergy.host</code>	<code>[host]</code>	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., <code>http://synergy.server:8400</code>).
<code>scontrol.rep{n}.synergy.dbpath</code>	<code>[path]</code>	Absolute synergy database path (e.g., <code>\\host\db\name</code>).
<code>scontrol.rep{n}.synergy.projspec</code>	<code>[specification]</code>	Synergy project specification which contains project name and its version (e.g., <code>name-version</code>).
<code>scontrol.rep{n}.synergy.login</code>	<code>[login]</code>	Synergy user name.
<code>scontrol.rep{n}.synergy.password</code>	<code>[password]</code>	Synergy password (not encoded).
<code>scontrol.rep{n}.synergy.port</code>	<code>[port]</code>	Synergy port.
<code>scontrol.rep{n}.synergy.remote_client</code>	<code>[client]</code>	(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.
<code>scontrol.rep{n}.synergy.local_dbpath</code>	<code>[path]</code>	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
<code>scontrol.rep{n}.type</code>	<code>tfs</code>	TFS repository type identifier.
<code>scontrol.rep{n}.tfs.url</code>	<code>[url]</code>	URL to TFS repository, e.g., <code>http://localhost:8080/tfs</code>
<code>scontrol.rep{n}.tfs.login</code>	<code>[login]</code>	TFS user name.
<code>scontrol.rep{n}.tfs.password</code>	<code>[password]</code>	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.ssdire	[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.

Visual Studio Settings

Setting Name	Value	Description
dottest.configuration		Configuration to be used during analysis when not specified with the <code>-config</code> switch is not used. See "Specifying Test Configurations", page 15
dottest.build.builder_id	msbuild visualstudio	Specifies which builder to use.
dottest.devenv.2015.path	Path to Visual Studio 2015 devenv.exe file	Allows users to customize the default path of Visual Studio 2015.
dottest.devenv.2013.path	Path to Visual Studio 2013 devenv.exe file	Allows users to customize the default path of Visual Studio 2013.
dottest.devenv.2012.path	Path to Visual Studio 2012 devenv.exe file	Allows users to customize the default path of Visual Studio 2012.
dottest.devenv.2010.path	Path to Visual Studio 2010 devenv.exe file	Allows users to customize the default path of Visual Studio 2010.
dottest.devenv.2008.path	Path to Visual Studio 2008 devenv.exe file	Allows users to customize the default path of Visual Studio 2008.
dottest.devenv.2005.path	Path to Visual Studio 2005 devenv.exe file.	Allows users to customize the default path of Visual Studio 2005.
dottest.visualstudio.version	2005 2008 2010 2012 2015	Allows users to override the default Visual Studio version automatically detected from the <code>*.sln</code> file. Uses the appropriate version of Visual Studio for building the solution regardless of <code>*.sln</code> files version.

Setting Name	Value	Description
<code>dottest.visualstudio.target</code>	Build Rebuild Deploy	Allows users to customize the target name used by Visual Studio during solution build. See http://msdn.microsoft.com/en-us/library/vstudio/xee0c8y7.aspx for details.
<code>dottest.visualstudio.custom_switches</code>		Allows users to pass additional switches to the <code>devenv.exe</code> command line during builds. See http://msdn.microsoft.com/en-us/library/vstudio/xee0c8y7.aspx for details.
<code>dottest.visualstudio.timeout</code>	Number of seconds for the timeout	Allows users to apply timeout during solution build (default is 3 hours).

Integrations

- Integrating with MSBuild
- Integrating with NAnt
- Integrating with Source Control Systems
- Using DTP Engines in an IDE
- Integrating with CI Tools

Integrating with MSBuild

Static Analysis Engine for .NET ships with built-in support for integration with MSBuild to simplify running it in MSBuild build scripts environments. Integration with MSBuild scripts is achieved with the following custom MSBuild task:

```
Parasoft.DotTest.MSBuild.Tasks.dll
```

Use the following code in your MSBuild script after a task is deployed:

```
<Import Project="$(MSBuildExtensionsPath)\Parasoft\Parasoft.DotTest.targets"/>

<Target Name="Demo">
  <DotTest
    Configuration="builtin://Demo"
    Solutions="C:\Devel\FooSolution\FooSolution.sln"
    Report="C:\Devel\Report"
    Out="C:\Devel\Out.txt" />
</Target>
```

Target File

The target file must be imported in the MSBuild script. If you use `deploy.exe`, then you can the following import statement:

```
<Import Project="$(MSBuildExtensionsPath)\Parasoft\Parasoft.DotTest.targets"/>
```

Alternatively, you can import the target directly from the installation directory:

```
<Import
  Project="[INSTALL_DIR]\integration\MSBuild\Parasoft.DotTest.targets\Parasoft.DotTest.targets"
/>
```

Targets can be directly imported from the installation directory when running multiple Static Analysis Engine for .NET installations on a single machine.

MSBuild Task

Use `<DotTest>` task in your build file to run Static Analysis Engine for .NET. The following arguments are supported:

- **Configuration:** Defines configuration used during analysis. See “Specifying Test Configurations”, page 15.
- **Solutions:** Defines solutions that are analyzed. See “Analyzing Visual Studio Solutions”, page 13.
- **Projects:** Defines projects that are analyzed. See “Analyzing Visual Studio Projects Without Solutions”, page 13.

- `Websites` : Defines web sites that are analyzed. See “Analyzing Websites”, page 13.
- `Resources`: Defines resources that are analyzed. See “Defining Test Scope”, page 18.
- `Report`: Defines path to report. See “Reporting”, page 32.
- `Settings`: Path to settings file. “Customizing DTP Engines for .NET”, page 52.
- `NoBuild`: Disables build of the tested solutions or projects.
- `SolutionConfig`: Solution build architecture. See “Specifying Solution Configuration and Target Platform”, page 88.
- `TargetPlatform`: Solution build architecture. See “Specifying Solution Configuration and Target Platform”, page 88.
- `Out`: Path where console output is saved.
- `DottestPath`: Path to `dottestcli.exe` file. Allows users to override the auto-detected `dottestcli.exe` path. This can be used to support multiple Static Analysis Engine for .NET installations on a single machine.

Integrating with NAnt

Static Analysis Engine for .NET ships with built-in support for integration with NAnt to simplify running it in NAnt build scripts environments. Use the following code in your NAnt script after the task is deployed to NAnt:

```
<target name="analyze">
  <dottest
    config="builtin://Demo"
    report="C:\Devel\Report"
    out="C:\Devel\Out.txt">
    <solutions>
      <include name="C:\Devel\FooSolution\FooSolution.sln">
    </solutions>
  </dottest>
</target>
```

Loading NAnt Task Library

Integration with NAnt scripts is achieved with the following custom NAnt task:

```
Parasoft.DotTest.NAnt.Tasks.dll.
```

This library must be in the same directory as `NAnt.exe` for the NAnt scripts to detect the Static Analysis Engine task. Alternatively, `<loadtasks>` can be used, which is useful for running multiple Static Analysis Engine for .NET installations on one machine:

```
<project name="sampleProject" default="test">
  <target name="test">
    <loadtasks>
      <fileset>
        <include
name=" [INSTALL_DIR] \integration\NAnt\Parasoft.DotTest.NAnt.Tasks.dll" />
      </fileset>
    </loadtasks>
  </target>
</project>
```

Supported NAnt Task Arguments

- `config`: Defines configuration used during analysis. See "Specifying Test Configurations", page 15.

- **solutions**: Defines solutions that are analyzed. List wildcards in ANT-style format separated by semi-colons to analyze two or more solutions:

```
<dottest config="builtin://Demo"
solutions="C:\Devel\FooSolution\FooSolution.sln;C:\Devel\Bar\**\*.sln" />
```

You can also nest `<include>` elements that point to ANT-style wildcards inside `<solutions>` elements:

```
<dottest config="builtin://Demo" >
  <solutions>
    <include name="C:\Devel\FooSolution\FooSolution.sln">
    <include name="C:\Devel\Bar\**\*.sln">
  </solutions>
</dottest>
```

See “Analyzing Visual Studio Solutions”, page 13.

- **projects**: Defines which projects are analyzed. List wildcards in ANT-style format separated by semi-colons to analyze two or more projects:

```
<dottest config="builtin://Demo"
projects="C:\Devel\FooProjects\Qux\Qux.csproj;C:\Devel\BarProjects\**\*.csproj" />
```

You can also nest `<include>` elements that point to ANT-style wildcards inside `<projects>` elements:

```
<dottest config="builtin://Demo" >
  <projects>
    <include name="C:\Devel\Foo\Qux\Qux.csproj">
    <include name="C:\Devel\Bar\**\*.csproj">
  </projects>
</dottest>
```

See “Analyzing Visual Studio Projects Without Solutions”, page 13.

- **websites**: Defines which web sites are analyzed. List wildcards in ANT-style format separated by semi-colons to analyze two or more web sites:

```
<dottest config="builtin://Demo"
websites="C:\Devel\Foo\WebSite;C:\Devel\Bar\*.WebSite" />
```

You can also nest `<include>` elements that point to ANT-style wildcards inside `<websites>` elements:

```
<dottest config="builtin://Demo" >
  <websites>
    <include name="C:\Foo\WebSite">
    <include name="C:\Bar\*.WebSite">
  </websites>
</dottest>
```

See “Analyzing Websites”, page 13.

- `resources`: Defines which resources are analyzed. Separate two or more paths to resources with a semi-colon:

```
<dottest config="builtin://Demo" resources="Foo/Bar/Baz;Foo/Qux/Garply" >
```

You can also nest `<res>` elements that point to paths inside `<resources>` elements:

```
<dottest config="builtin://Demo">
  ...
  <resources>
    <res name="Foo/Bar/Baz" />
    <res name="Foo/Qux/Garply" />
  </resources>
</dottest>
```

See “Defining Test Scope”, page 18.

- `report`: Defines path to the report.

```
<dottest
  config="builtin://Demo"
  report="C:\Foo\Report" >
  ...
</dottest>
```

See “Reporting”, page 32.

- `settings`: Defines path to settings file:

```
<dottest
  config="builtin://Demo"
  settings="C:\Foo\settings.properties" >
</dottest>
```

See “Customizing DTP Engines for .NET”, page 52.

- `nobuild`: Disables build of the tested solutions or projects.

```
<dottest
  config="builtin://Demo"
  nobuild="true" >
  . . .
</dottest>
```

- `solutionConfig`: **Solution build architecture.** See “Specifying Solution Configuration and Target Platform”, page 88.
- `targetPlatform`: **Solution build architecture.** See “Specifying Solution Configuration and Target Platform”, page 88.
- `Out`: **Path where console output is saved.**
- `DottestPath`: **Path to `dottestcli.exe` file.** Allows users to override the auto-detected `dot-testcli.exe` path. This can be used to support multiple Static Analysis Engine for .NET installations on one machine.

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 dotTEST 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 TeamCity

Team City is a continuous integration build system where Web Service may be installed on different machine than its builds agents.

Integrating DTP Engines with TeamCity

1. Install DTP Engines for .NET on a machine or machines where the Team City build agent is installed.
2. Perform one of the following actions:
 - Choose "Team City" on the "Select Build System for Integration" tab during installation and follow the wizard instructions.
 - Run the following command manually and follow the wizard instructions:


```
[INSTALL_DIR]\Integrators\TeamCity\deploy.exe
```
3. If TeamCity does not automatically recognize and deploy its plug-in to the build agent, restart TeamCity server and build agent services in **Control Panel> Administrative Tools> Services**.

Configuring a TeamCity Project for DTP Engines

1. Launch TeamCity web services.
2. Choose **Projects> [Project Name]**.
3. Click **Edit Project Settings**.
4. Create a build configuration or edit an existing configuration.
5. At configuration step 3 under "Build Steps", add a build step.
6. Choose Parasoft dotTEST runner.
7. Complete fields on the form.
8. Save the configuration.
9. Ensure that at least one agent is capable of running the configuration (see Step 8 "Agent Requirements").
10. Run the Configuration.

Viewing Results

1. Choose **Projects> [Project Name]**.
2. Go to the Configuration that was used for the run.
3. Click **Build Log** to view the logs.
4. Click **Artifacts** to view the report.

Integrating with Jenkins

DTP Engines for .NET 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 from GitHub, see [Parasoft Findings Plugin Project](#). If you need additional information on how to rebuild the plugin, contact Parasoft Support.

Building Solutions and Projects

Many Static Analysis Engine for .NET features require source code to be compiled into binaries. By default, Static Analysis Engine attempts to build solutions and projects prior to analyzing them.

Delegating the Build to MSBuild or Visual Studio

Static Analysis Engine for .NET delegates the build to `msbuild.exe` by default when Visual Studio is not installed on the machine. If an appropriate version of Visual Studio is present, the build is performed by the `devenv.exe /Build` mechanism.

Define the `dottest.build.builder_id` setting to either `visualstudio` or `msbuild` to explicitly set Visual Studio or MSBuild as the builder. See “Settings Reference”, page 54, for additional information.

Depending on Pre-built Code

Pass the `-nobuild` switch to skip the building phase during analysis if the code is compiled prior to analysis.

Specifying Solution Configuration and Target Platform

If code is built during or prior to analysis, Static Analysis Engine for .NET needs to know the correct Solution Configuration (e.g. Debug, Release, or other) and Target Platform (e.g. Any CPU, x86, or other). Static Analysis Engine attempts to choose the most suitable one, but it is recommended to specify them explicitly.

You can use the `-solutionConfig` and `-targetPlatform` command line switches to specify these Solution Configuration and Target Platform. See “Switches Reference”, page 91, for additional information.

These can also be set with the `dottest.build.solution_config` and `dottest.build.target_platform` settings in the `dottestcli.properties` configuration file.

Verifying the Required Build Artifacts

Prior to analysis, DTP Engines for .NET need to verify that all required build artifacts, such as `.exe`, `.dll` or `.pdb` files, are available. Artifacts may not be available due to build issues or incorrect setup, which may prevent DTP Engines from performing analysis or collecting complete analysis results.

If DTP Engines are unable to find all required `.exe`, `.dll` or `.pdb` files, the missing artifacts will be listed in the Setup Problems section of your report.

Getting Help

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

```
dottestcli.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 colon and the backslashes to specify the package location. The following example shows what the path may look like:

```
C:\\Project\\Mailssystem\\Report
```

You can also collect logs manually from the following location (the domain name may need to be replaced with the machine name):

```
%ProgramData%\Parasoft\dotTEST\<<Domain>_<User>\logs
```

If you run dotTEST DTP Engine with Visual Studio, additional logs can be collected form:

```
%localappdata%\Parasoft\DtpPlugin\logs\
```

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

Switches Reference

The following table describes the switches available in DTP Engines

Switch	Description
-solution PATH	Path to solution(s) to be analyzed. Specify multiple times to analyze many solutions. Supports ANT-style wildcards.
-project PATH	Path to project(s) to be analyzed when solution is not provided. Specify multiple times to analyze many projects. Supports ANT-style wildcards
-website DIR	Full path to web site directory to be analyzed when solution is not provided.
-config CONFIG_URL	Path to test configuration. Can point to built-in, user-defined, or team configurations stored in DTP. Can reference by filename or by HTTP URL. See “Specifying Test Configurations”, page 15.
-resource RESOURCE	Solution path of a resource(s).The path corresponds to path of an element in Solutions Explorer of Visual Studio. See “Defining Test Scope”, page 18.
-include PATH	File-system paths of files to include in the analysis. Supports ANT-style wildcards. If not specified, all files are analyzed. See “Fine-tuning the Scope”, page 18
-exclude PATH	File-system paths of files to exclude from the analysis. Supports ANT-style wildcards. See “Fine-tuning the Scope”, page 18.
-report PATH	Path to report directory or main report file. See “Reporting”, page 32.
-publish	Publishes report to Development Testing Platform (DTP) server. “Sending Results to Development Testing Platform (DTP) Server”, page 38.
-settings PATH	Path to report directory or main report file. See “Customizing DTP Engines for .NET”, page 52.
-nobuild	Disables build of the tested solution or projects.
-solutionConfig CONFIG	Solution configuration, e.g. Debug.
-projectConfig CONFIG	Project configuration, e.g. Debug.
-targetPlatform PLATFORM	Solution configuration target platform, e.g. Any CPU; or project configuration target platform, e.g. AnyCPU. Note the syntax difference.
-out PATH	Path where console output is saved.
-help	Displays command line help.
-version	Prints version
-listconfigs	Prints all available test configurations.
-encodepass PASSWORD	Prints an encoded password that can be used in the .properties configuration file.

Switch	Description
-reference PATH	Path to additional assemblies needed to resolve dependencies of the analyzed projects. Use this switch if you receive an "Unable to find reference assembly" message.
-runtimeCoverage PATH	Path to the file that contains runtime coverage data
-staticCoverage PATH	Path to the file that contains static coverage data
-machineID	Prints your machine ID.
-showsettings	Prints the current settings and customizations.

Third-Party Content

DTP Engines for .NET 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
fluent-hc.jar	Apache License 2.0
httpclient.jar	Apache License 2.0

Item	License
httpcore.jar	Apache License 2.0
httpclient-cache.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
Java JRE	Oracle Binary Code License
Microsoft Visual C++ Redistributable Libraries	Microsoft Visual C++ Redistributable Libraries (Visual Studio License)
IKVM	Permissive Free Software Licence
Metaspec C# Parser	Metaspec Commercial License
log4cpp	GNU LGPL License
.NET Compiler Platform ("Roslyn")	Apache License 2.0
DotNetZip	Microsoft Public License
Osgi.Framework	Apache License 2.0
Apache Felix Declarative Services	Apache License 2.0
Java Native Access (JNA)	GNU LGPL License
NUnit	The zlib/libpng License (Zlib)
Microsoft.Composition	Microsoft Software License Terms
Microsoft.Tpl.Dataflow	Microsoft Software Supplemental Licence Terms
Nito AsyncEx	Microsoft Public License
System.Collections.Immutable	Microsoft Software License Terms
System.Reflection.Metadata	Microsoft Software License Terms

Item	License
Common Compiler Infrastructure	Microsoft Public License