Testing from the Command Line Interface

This topic explains how to run a test from the C++test command line interface (`cpptestcli`), which is described in Command Line Interface (cli).

Sections include:

Prerequisites

The command line mode requires a command line interface license (available with C++test Server Edition).

Extended Command Line Mode vs. Desktop Command Line Mode

There are two command line interface licenses available for C++test:

1. **Extended Command Line Mode** is provided in Server Edition and Server Edition IT, and available for Custom Editions.
2. **Desktop Command Line Mode** is available for Custom Editions. The Desktop Command Line Mode provides similar functionality to the Extended Command Line Mode, except that parallel processing is limited to simultaneously executing 8 parallel threads for a given task (e.g. static analysis) in the Desktop Command Line Mode.

- To access the full functionality available with the Server Edition, you also need to install and configure Parasoft Team Server.
- We strongly recommend that you configure C++test preferences (for Team Server, task assignment, reporting, etc.) and team Test Configurations as described in the Configuration before you start testing.
- For command line execution, you will need to ensure that the installation directory is on the path, or launch cpptest with the full path to the executable (for example, `c:\parasoft\c++test\cpptestcli.exe`).Before you can test code with C++test, it must be added to an Eclipse C /C++ project. For instructions on creating a new project, see Creating a Project.

- Before you perform the initial test, we strongly recommend that you review and modify project options. For details on how to do this, see Local Settings (Options) Files.
- For `cpptestcli` to email each developer a report that contains only the errors/results related to his or her work, one of the following conditions must be true:
    - You have configured C++test to compute code authorship based on source control data AND your project is under a supported source control system AND each developer's source control username + the mail domain (specified using an options file and the `-localsettings` option described in `-localsettings %LOCALSETTINGS_FILE%`) matches the developer's email address.
    - You have configured C++test to compute code authorship based on local user AND each user name + the mail domain (specified using an options file and the `-localsettings` option described in `-localsettings %LOCALSETTINGS_FILE%`) matches the developer's email address.

Setup Overview

Parasoft C++test has two user modes: interactive desktop usage in the GUI and command line mode via the command line interface (CLI). The CLI interface is a standard feature of the Server Edition.

CLI mode is typically used to perform regular or continuous code analysis and test in conjunction with regular/continuous builds or as a part of an automatic regression test infrastructure. C++test CLI can be invokedon the specified project resourcesAs part of the CLI execution, C++test can perform one or more of the following:

- Static analysis of code, including checks against a configured coding policy, analysis of possible runtime bugs, and metrics analysis.

- Execution of unit tests

- Analysis of SCM code repository to identify code changes since the last run and initiate code review sessions on updated code.
- Generation of reports and their distribution to a central report server and/or to individual developers and managers, according to specified reporting configurations.

As part of the execution, C++test can use your SCM client (if supported) to automatically retrieve file modification information from the SCM system and generate tasks for specific individuals based on results of code analysis and executed tests.

Specific execution options for C++test are controlled via Test Configurations and Preferences.

Test Configurations can be sourced from the built in set, or created using C++test interactive mode in the GUI. It is highly recommended that you do not use the built-in configurations (other than for getting started). We suggest using the built in configurations as starting templates for customer-specific configurations, which are then stored on disk or on Parasoft Team Server.

Preferences can be configured from the C++test GUI. Most of the preference settings can also be supplied with a configuration file that is provided as a parameter to a CLI call. A table of the configuration file preference settings is available in Local Settings (Options) Files. C++test preferences set from the GUI are applied by default. These can be overridden — on an individual basis—by preference values contained in the configuration file used with a given run. This enables you to have a basic set of preferences configured for all CLI runs, and then vary individual settings as necessary by providing an additional configuration file for a specific run with a given Test Configuration. This can be useful, for example, to include different information in reports for different runs, or to change options for email distribution of reports, including report names, email headings, etc.

Step 1: Configure Preferences

C++test preferences and Parasoft Test preferences (which apply across Parasoft products) are accessed through the **Parasoft> Preferences** menu. Start by configuring the following preferences:

- **License**: Specify the license or License Sever settings.
- **Team**: Check **Enable Team Server**. If Team Server is not autodetected, enter the Team Server's IP address in **Server Information> Host Name**. If you are running Team Server on the same machine as your Server Edition product, enter `localhost` . Unless you changed the Team Server default port (18888) when it was installed, do not change the port here. Click **Test Connection** to verify the correct settings.
- **Source Controls**: These settings enable automatic mapping of the tool results to the individuals who last changed the affected code or test artifact. Check your source control system, and use the instructions in Connecting to Source Control to set the options appropriate for your SCM.
- **Scope and Authorship**: Check the appropriate options for your environment as described in Configuring Task Assignment and Code Authorship Settings.
- **Reports**: The following options are enabled by default and are a good starting point:
    - **Detailed report for developers** (includes task breakdown with details).
    - **Overview of tasks by authors** (summary table).
    - **Generate formatted reports in command line mode**.
    - **Suppressions Details** (applies to static analysis only).
- **E-mails**: Enter settings that will be used to send emails with reports. This needs to be an existing email account on an email server accessible from the C++test test machine.
- **Reports> Email Notifications**:
    - If desired, enable **Send Reports by Email**. Regardless of this setting, reports will always be uploaded to Parasoft Team Server for later viewing (controlled by the CLI option). Email distribution will use the settings for E-mails above.
    - Manager reports contain a rollup of all test results generated by C++test Developer reports contain only results for individual developers. Enable options and specify email addresses accordingly.

Step 2: Customize Test Configurations

Create a custom Test Configuration as described in Configuring Test Configurations and Rules for Policies. See Configuring Test Configurations for details on C++test-specific options.

Step 3: Create a localsettings File

Create a localsettings file as described in Local Settings (Options) Files.

Step 4: Activate CLI in the Currently-Running Build System (e.g., batch script)

For example, a sample command line to be added might be:

- `cpptestcli -data "c:\MyWorkspace" -resource "ProjectToTest" -config builtin://ShouldHaveRules -publishteamserver -localsettings acme_policy.settings`

The reports will be sent after each batch run, and trend reports will be populated with data. The reports will also be available for viewing via **Parasoft> Explore> Team Server Reports**.

cli Usage

The general procedure for testing from the command line is as follows:

- Use the `cpptestcli` utility, with appropriate options, to launch analysis in the command-line mode. A complete list of options is provided in cli Options. Key options are:
    - **-data:** Specifies the Eclipse workspace location.
    - **-config:** Specifies Test Configuration.
    - **-resource:** Specifies the resource (e.g., project, folder, file) to be tested.
    - **-publish:** Publishes test results to DTP.
    - **-publishteamserver**: Publishes test results to Team Server.
    - **-report:** Generates a report.
    - **-localsettings:** Passes advanced settings for Team Server/Parasoft DTP/mail reporting. Options are described in Local Settings (Options) Files.

Testing Headers

C++test does not directly test headers unless they are included by a source file under test. See How do I analyze header files/what files are analyzed? for details.

Testing Template Functions

C++test does perform static analysis and unit testing of instantiated function templates and instantiated members of class templates. See Support for Template Functions for details.

Notes for Command Line Testing on Windows

- C++test does not support file paths specified using Cygwin's `"/cygdrive/DISK/PATH"` format; instead, use the standard Windows path format.
- Depending on the shell/console, backslashes in file paths should be escaped/doubled; e.g.,`"C:\\MyLocation\\MyFile"`
- All backslashes in file paths must be escaped/doubled when used in options files (with the `-localsettings` option). Alternatively, you can use forward slashes; e.g., `"C:/MyLocation/MyFile".`

cli Invocation

The general form of invocation for `cpptestcli` is:

- `cpptestcli [OPTIONS]`

Typically, invocations follow this pattern:

- `cpptestcli -data "c:\MyWorkspace" -resource "ProjectToTest" -config builtin://ShouldHaveRules -publish`

Excluding Specific Project Resources from Analysis/Testing

If you want to exclude some files from analysis/testing (for instance, to prevent static analysis of automatically-generated files), you can indicate which project resources should not be tested as described in Excluding Project Resources from Testing. Perform this configuration in the GUI, then the settings will be applied for all tests on this project—from the GUI or from the command line.

Using -data to Specify Your Eclipse Workspace

If you are not in the same directory as the Eclipse workspace that you want to test, you need to use `cpptestcli` with the `-data` option. For example, this Windows command tests the C++test Example project by applying the "My Configuration" Test Configuration, generates a report of results, and saves that report in the `c:\reports\Report1` directory

```
cpptestcli -data "c:\Documents and Settings\cynthia\ApplicationData\Parasoft\C++test\workspace" -resource
"C++test Example" -config user://"My Configuration" -report c:\reports\Report1
```

If you are in the same directory as the workspace that you want to test, you can call `cpptestcli` without the`-data` option. For example, this Windows command tests the C++test Example project by applying the My Configuration Test Configuration, generates a report of results, and saves that report in the `c:\reports\Report1` directory:

```
cpptestcli -resource "C++test Example" -config user://"My Configuration" -report c:\reports\Report1
```

cli Options

Available `cpptestcli` options are listed in the following tables.

General Options

- `-config %CONFIG_URL% -` Specifies that you want to run the Test Configuration available at `%CONFIG_URL%.`
  This parameter is required except when importing projects. `%CONFIG_URL%` is interpreted as a URL, the name of a Test Configuration, or the path to a local file. Examples:
  - By filename:
    `-config "mylocalconfig.properties"`

    - By URL:
      `-config "http://intranet.acme.com/cpptest/team_config.properties"`
    - Built-in configurations:
      `-config "builtin://Demo Configuration"`
      `-config "Demo Configuration"`
    - User-defined configurations:
      `-config "user://My First Configuration"`
    - Team configurations:
      `-config "team://Team Configuration"`
      `-config "team://teamconfig.properties"`
- `-help` - Displays help information. Does not run testing.
- `-machineid` - Prints your machine ID.
- `-localsettings %LOCALSETTINGS_FILE% -` Reads the options file `%LOCALSETTINGS_FILE%` for global preferences. These settings specify details such as Parasoft DTP settings, email settings, and Team Server settings.
  The options file is a properties file. These files can control reporting preferences (who should reports be sent to, how should those reports be labelled, what mail server and domain should be used, etc.) Team Server settings, Parasoft DTP settings, email settings, and more. For details on creating options files; see Local Settings (Options) Files.
- `-nobuild` - Prevents C++test from rebuilding the project before testing it. Use this option if the project is already built before the test run.
- `-fail` - Fails the build by returning a non-zero exit code if any violations are reported.
- `-publish` - Publishes the report to DTP. You can enable sending reports to DTP in the GUI or in the command line mode; see Connecting to DTP.
- `-publishteamserver` - Publishes the report to the Team Server. The Team Server location can be specified in the GUI or in the options file (described in the `-localsettings %LOCALSETTINGS_FILE%` entry).
- `-report %REPORT_FILE% -` Generates an XML report to the given file `%REPORT_FILE%` and adds an HTML (or PDF or custom format—if specified using the report.format option) report with the same name (and a different extension) in the same directory.
  All of the following commands will produce an HTML report `filename.html` and an XML report `filename.xml`.
    - `-report filename.xml`
    - `-report filename.htm`
    - `-report filename.html`

  If the specified path ends with an ".html"/".htm"/".xml" extension, it will be treated as a path to the report file to generate. Otherwise, it will be treated as a path to a directory where reports should be generated.
  If the file name is explicitly specified in the command and a file with this name already exists in the specified location, the previous report will be overwritten. If your command doesn't explicitly specify a file name, the existing report file will not be overwritten—the new file will be named repXXXX.html, where XXXX is a random number.
  If the `-report` option is not specified, reports will be generated with the default names "report.xml/html" in the current directory.
- `-dtp.autoconfig %PROJECT_NAME@SERVER_NAME:port% -` Pulls settings stored on the DTP server (recommended for ease of maintenance — especially if you do not already have a locallocally stored settings file).
  For example:
  `-dtp.autoconfig Project1@dtp.company.com:8080`
- `-encodepass <plainpassword> -` Generates an encoded version of a given password. Prints the message 'Encrypted password: <encpass>' and terminates the cli application.
  If your nightly process will 1) login to Team Server and b) send emails, you can use this option to encrypt the required passwords.
- `-showdetails` - Prints detailed test progress information.
- `-buildscript %SCRIPT_FILE%` - Executes the specified build script prior to any testing. See Using the cli with an Eclipse-Based Builder.

- `-appconsole stdout|% OUTPUT_FILE%` - Redirects C++test's console output to standard output or an `%OUTPUT_FILE%` file.
  Examples:
  `-appconsole stdout` (console redirected to the standard output)
  `-appconsole console.out` (console redirected to console.out file)
- `-list-compilers` - Prints a list of valid compiler family values.
- -list-configs - Prints a list of valid Test Configuration values.
- `-include %PATTERN%, -exclude %PATTERN%` - Specifies files to be included/excluded during testing.
  You must specify a file name or path after this option.
  Patterns specify file names, with the wildcards *and ? accepted, and the special wildcard ** used to specify one or more path name segments.
  Syntax for the patterns is similar to that of Ant filesets.
  Examples:
  `-include **/Bank.cpp` (test Bank.cpp files)

  `-include **/ATM/Bank/*.cpp` (test all .cpp files in folder ATM/Bank)

  `-include c:/ATM/Bank/Bank.cpp` (test only the c:/ATM/Bank/Bank.cpp file)

  `-exclude **/internal/**` (test everything except classes that have path with folder "internal")

  `-exclude **/*Test.cpp` (test everything, but files that end with Test.cpp)

  Additionally if a pattern is a file with a .lst extension, it is treated as a file with a list of patterns.
  For example, if you use -include c:/include.lst and include.lst contains the following (each line is treated as single pattern):
  **/Bank.cpp

  `**/ATM/Bank/*.cpp`

  `c:/ATM/Bank/Bank.cpp`

  then it has same effect as specifying:
  `-include **/Bank.cpp -include **/ATM/Bank/*.cpp`

```
-include c:/ATM/Bank/Bank.cpp"
```

Options for Importing and Creating Projects

Option	Purpose	Notes

`-import %ECLIPSE_PROJECT%`	Imports the specified Eclipse project(s) into the Eclipse workspace.	If `%ECLIPSE_PROJECT%` is a .project file, the selected project will be imported
		
		If it is a directory, all Eclipse projects found in the selected directory and subdirectories will be imported.
		
		Examples:
		
		`-import \".project\"`
		
		`-import \"c:\\DevelRootDir\"`
		
		The `-config` option is not necessary while using -import. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.

`-bdf <cpptestscan.bdf>`	Creates C++test projects from build data files (.bdf). To prepare a build data file, perform a build of the project with the `cpptestscan` utility as the prefix for the compiler / linker executable.	Example: `-bdf "cpptestscan.bdf"` See Creating a Project Using an Existing Build System for details. Options can be specified in the options file. See Local Settings (Options) Files for details. The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.
-ccs %CCS_PROJECT%	Imports TI Code Composer Studio projects, If `%CCS_PROJECT%` is: pjt project file - the selected project will be imported directory - all .pjt projects found in selected directory and subdirectories will be imported	Examples: -ccs "MyProject.pjt" -ccs "c:\DevelRootDir" The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.
`-dsp <.dsp file | .dsw file | root location>`	Creates C++test projects from Microsoft Visual Studio projects. Specify a 6.0 project file (.dsp), Microsoft Visual Studio 6.0 workspace file (.dsw), or root directory.	Visual Studio 6.0 project import options can be specified in the options file. See Local Settings (Options) Files for details. The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.
`-ewp %EWP_PROJECT%`	Imports IAR Embedded Workbench projects. If `%EWP_PROJECT%` is: .ewp project file - the selected project will be imported .eww workspace file - all projects from the workspace will be imported directory - all .ewp projects found in selected directory and subdirectories will be imported	Examples: -ewp "MyProject.ewp" -ewp "MyWorkspace.eww" -ewp "c:\DevelRootDir" The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.

`-gpj <. prj_root_file>`	Creates C++test projects from Green Hills .gpj projects.	Green Hills .gpj project import options can be specified in the options file. See Local Settings (Options) Files for details. The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed.
`-hew %HEW_PROJECT%`	Imports Highperformance Embedded Workshop projects. The following can be specified as `%HEW_PROJECT%`: **.hwp project file**: The selected project will be imported. **.hws workspace file**: All projects from the workspace will be imported. **directory**: All .hwp projects found in the selected directory and subdirectories will be imported	The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed. Examples: -hew "MyProject.hwp" -hew "MyWorkspace.hws" -hew "c:\DevelRootDir"
`-uv %KEILUV_PROJECT%`	Imports Keil uVision3 projects. If `%KEILUV_PROJECT%` is: .uv2 project file - the selected project will be imported directory - all .uv2 projects found in selected directory and subdirectories	The `-config` option is not necessary while using `-import`. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed. Examples: -uv "MyProject.uv2" -uv "c:\DevelRootDir"
-vcp %VCP_PROJECT%	Imports Microsoft eMbedded Visual C++ 4.0 projects. If `%VCP_PROJECT%` is: .vcp project file - the selected project will be imported .vcw workspace file - all projects from the workspace will be imported directory - all .vcp projects found in selected directory and subdirectories will be imported	The `-config` option is not necessary while using -import. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed. Examples: -vcp "MyProject.vcp" -vcp "MyWorkspace.vcw" -vcp "c:\DevelRootDir"
`-wpj %WPJ_PROJECT%`	Imports Wind River Tornado project. If `%WPJ_PROJECT%` is: the .wpj project file - selected project will be imported .wsp workspace file - all projects from the workspace will be imported directory - all .wpj projects found in selected directory and subdirectories will be imported	The `-config` option is not necessary while using -import. If the `-config` option is specified, then the workspace with the imported project(s) will be tested; otherwise, the project will be imported, but no testing will be performed. Examples: -wpj "MyProject.wpj" -wpj "MyWorkspace.wsp" -wpj "c:\DevelRootDir"

Options for Testing Projects Available in the C++test/Eclipse Workbench

Option	Purpose	Notes
`-data %WORKSPACE_DIR%`	Specifies the location of the Eclipse workspace directory to use.	Defaults to the current user's dependent directory.

`-resource %RESOURCE%`	Specifies the path to the workspace resource %RESOURCE% to test.	Use multiple times to specify multiple resources.
		
		Use quotes when the resource path contains spaces or other non-alphanumeric characters.
		
		If %RESOURCE% is a .properties file, the value corresponding to `com.parasoft.xtest.checkers.resources` will be interpreted as a colon(:)-separated list of resources. Only one properties file can be specified in this way. If %RESOURCE% is a `.lst` file, each line will be treated as a resource. If no resources are specified on the command line, the complete workspace will be tested.
		
		Team Project Set File (PSF) files are supported for CVS, SVN, Star Team, and other source control systems (depending on the Eclipse plugin capabilities installed).
		
		Paths (even absolute ones) are relative to the workspace specified by the -data parameter.
		
		Examples:
		
		`-resource "Acme Project"`
		
		`-resource "/MyProject/src/com/acme/MyClassTest.java"`
		
		`-resource "/MyProject/src/com/acme"`
		
		`-resource testedprojects.properties`

Notes

- To see a list of valid command line options, enter for `cpptestcli -help`.
- `cpptestcli` automatically emails designated group managers and architects a report that lists all team/project errors and identifies which developer is responsible for each error. If no errors are reported, reports will be sent unless the options file contains the `report.mail.on.error.only=true` option.
- If the appropriate prerequisites are met, `cpptestcli` automatically emails each developer a report that contains only the errors /results related to his or her work. If no errors are reported for a particular developer, a report will not be emailed to that developer.

Local Settings (Options) Files

Localsettings files can be passed at the command line to control options for reporting, task assignment, licensing, and more. This lets you:

- Configure and use different setting configurations for different projects.
- Extend or override team-wide settings as needed (for example, for settings that involve local paths).
- Adjust settings without having to open the GUI.

Options files can control report settings, Parasoft DTP settings, error authorship settings, Team Server settings, and more. You can create different options files for different projects, then use the DTP settings, error authorship settings, Team Server settings, and more. You can create different options files for different projects, then use the
`-localsettings` option to indicate which file should be used for the current command line test.

Each options file must be a simple text file. There are no name or location requirements. Each setting should be entered in a single line.

If a parameter is specified in this file and there is an equivalent parameter in the GUI's Preferences panel (available from **Parasoft> Preferences**), the parameter set in this file will override the related parameter specified from the GUI. If a parameter is not specified in this file, C++test will use the equivalent parameter specified in the GUI.

Any options for creating or importing projects are valid *only when creating or importing the project*. They are ignored during subsequent runs.

Creating a Local Settings (Options) File by Exporting Your GUI Preferences

The fastest and easiest way to create options files is to export your Preferences from the GUI.

1. Choose **Parasoft> Preferences**.
2. Select **Parasoft** (the root element in the left tree).
3. Click the **share** link in the right side of the panel.
4. In the dialog that opens, specify which preferences you want to export to a file.
5. Click the **Browse** button, then specify the file where you want the settings saved.
6. Click **OK**.

    - If you select an existing file, the settings will be appended to that file. Otherwise, a new file will be created.
    - Exported passwords will be encrypted.

Options files can determine the following settings:

- Reporting Settings
- Parasoft DTP Settings
- Project Center Settings
- Team Server Settings
- Licensing Settings
- Technical Support Settings
- Authorship/Scope Settings
- Source Control Settings

- Settings for Creating BDF-Based Projects
- Settings for Importing Green Hills .gpj Projects
- Settings for Importing Microsoft Visual Studio 6.0 .dsp Projects

- Miscellaneous Settings

Notes

- Each setting should be entered on a single line.
- If your options file contains any invalid settings, details will be reported in the command line output.
- If your are running cli mode from a developer/tester desktop (as opposed to from a Server machine), use the `tasks.clear=false` option to ensure that your results from previous runs are preserved.

Reporting Settings

Setting	Purpose
`report.associations`	Specifies whether the report shows requirements, defects, tasks, and feature requests that are associated with a test.
`report.authors_details`	Determines whether the report includes an overview of the number and type of tasks assigned to each developer. The default is `true`.
`report.contexts_details`	Determines whether the report includes an overview of the files that were checked or executed during testing.The default is `false`.
`report.custom.extension` `report.custom.xsl.file`	Specifies the location and extension of the XSL file for a custom format. Used with `report.format=custom` For details and examples, see Configuring Reporting Settings.
`report.developer_errors=true|false`	Determines whether manager reports include details about developer errors.
`report.developer_reports=true|false`	Determines whether the system generates detailed reports for all developers (in addition to a summary report for managers).
`report.format=html|pdf|custom`	Specifies the report format.
`report.generate_htmls=true|false`	Determines whether HTML reports are generated and saved on the local file system. XML reports are generated and saved regardless of this setting's value. The default setting is `true`.
`report.graph.cs_start_date=[MM/dd/yy]`	Determines the start date for trend graphs that track static analysis tasks over a period of time. See Understanding Reports for more details on these reports.
`report.graph.ue_coverage_start_date=[MM/dd/yy]`	Determines the start date for trend graphs that track coverage over a period of time. See Understanding Reports for more details on these reports.

`report.location_details=true|false`	Specifies whether absolute file paths are added to XML data. This needs to be enabled on the Server installation if you want to relocate tasks upon import to desktop installations.
`report.mail.attachments=true|false`	Determines whether reports are sent as attachments. All components are included as attachments; before you can view an HTML report with images, all attachments must be saved to the disk. The default setting is `false`.
`report.mail.cc=[email_addresses]`	Specifies where to mail comprehensive manager reports. This setting must be followed by a semicolon-separated list of email addresses. This setting is typically used to send reports to managers or architects. It can also be used to send comprehensive reports to developers if developer reports are not sent automatically (for example, because the team is not using a supported source control system).
`report.mail.compact=trends|links`	Specifies that you want to email a compact report or link rather than a complete report. If `trends` is used, the email contains a trend graphs, summary tables, and other compact data; detailed data is not included. If `links` is used, the email contains only a link to a report (which is available on Team Server)
`report.mail.domain=[domain]`	Specifies the mail domain used to send reports.
`report.mail.enabled=true|false`	Determines whether reports are emailed to developers and to the additional recipients specified with the `cc` setting. Remember that each developer that worked on project code will automatically be sent a report that contains only the errors/results related to his or her work.
`report.mail.exclude=[email_addresses]`	Specifies any email addresses you do not want to receive reports. This setting is used to prevent C++test from automatically sending reports to someone that worked on the code, but should not be receiving reports.
`report.mail.exclude.developers=true|false`	Specifies whether reports should be mailed to any developer whose email is not explicitly listed in the `report.mail.cc` property. This setting is used to prevent reports from being mailed to individual developers.
`report.mail.format=html|ascii`	Specifies the email format.
`report.mail.from=[email_address OR user_name_of_the_same_domain]`	Specifies the "from" line of the emails sent.
`report.mail.include=[email_addresses]`	Specifies the email addresses of developers that you want to receive developer reports. This setting must be followed by a semicolon-separated list of email addresses. This setting is typically used to send developer reports to developers if developer reports are not sent automatically (for example, because the team is not using a supported source control system). It overrides developers specified in the 'exclude' list.
`report.mail.on.error.only=true|false`	Determines whether reports are sent to the manager only if an error is found or a fatal exception occurs. Developer emails are not affected by this setting; developer emails are sent only to developers who are responsible for reported errors. The default setting is `false`.
`report.mail.server=[server]`	Specifies the mail server used to send reports.
`report.mail.subject=My New Subject`	Specifies the subject line of the emails sent. The default subject line is "C++test Report." For example, if you want to change the subject line to "C++test Report for Project A", you would use `report.mail.subject=C++test Report for Project A`
`report.mail.time_delay=[server]`	Specifies a time delay between emailing reports (to avoid bulk email restrictions).
`report.mail.unknown=[email_address OR user_name_of_the_same_domain]`	Specifies where to mail reports for errors assigned to "unknown".

`report.mail.username=[username]`
`report.mail.password=[password]`
`report.mail.realm=[realm]`
	Specifies the settings for SMTP server authentication.
	
	The `realm` value is required only for those servers that authenticate using SASL realm.

`report.active_rules=true|false`	Determines if C++test reports contain a list of the rules that were enabled for the test.
`report.suppressed_msgs=true|false`	Determines whether reports include suppressed messages. The default setting is `false`.
`session.tag=[name]`	Specifies a session tag used to label these results. This value is used for uploading summary results to Team Server. The tag is an identifier of the module checked during the analysis process. Reports for different modules should be marked with different tags.
`report.ue_coverage_details_htmls=[coverage_type]`	Determines whether a test's HTML report links to another report that includes source code annotated with line-by-line coverage details. The following values can be used for `[coverage_type]`: FC - for function coverage LC - for line coverage SC - for statement coverage BCC - for block coverage DC - for decision coverage SCC - for simple condition coverage MCDC - for MC/DC coverage CC - for Call Coverage
`report.metrics_details=true|false`	Determines whether an XML report with metrics summary information (as well as individual class and method detail data where applicable) is produced. This report will be generated only when a metricsenabled Test Configuration is run.

Parasoft DTP/ Project Center Settings

Setting	Purpose
`dtp.enabled=true|false`	Determines whether the current C++test installation is connected to DTP. This setting is not needed if you want to use the value specified in the GUI.
`concerto.reporting=true|false`	Determines whether the current C++test installation is connected to Parasoft Project Center. This setting is not needed if you want to use the value specified in the GUI.
`dtp.autoconfig=true|false`	Enables autoconfiguration with C++test settings stored on the DTP server
`dtp.server=[server]`	Specifies the host name of the DTP server. This setting is not needed if this information is specified in the GUI.
`concerto.data.port=[port]`	Specifies the Parasoft Project Center port. This setting is not needed if you want to use the value specified in the GUI.
`dtp.port=[port]`	Specifies the port number of the DTP server. This setting is not needed if you want to use the value specified in the GUI.
`concerto.user_defined_attributes=[attributes]`	Specifies the user-defined attributes for Parasoft Project Center. Use the format `key1:value1; key2:value2` For more details on attributes, see Connecting to Project Center. This setting is not needed if you want to use the value specified in the GUI.
`concerto.log_as_nightly=true|false`	Determines whether the results sent to Parasoft Project Center are marked as being from a nightly build.

`concerto.use_resource_attributes=true|false`	Determines whether Parasoft Project Center attributes specified in the GUI at the project level should be used. This allows you to disable project level Parasoft Project Center attributes.
`dtp.project=[project_name]`	Specifies the name of the DTP project that you want these results linked to.

Team Server Settings

Setting	Purpose
`tcm.server.enabled=true|false`	Determines whether the current C++test installation is connected to the Team Server. This setting is not needed if you want to use the value specified in the GUI.
`tcm.server.name=[name]`	Specifies the machine name or IP address of the machine running Team Server. This setting is not needed if you want to use the value specified in the GUI.
`tcm.server.port=[port]`	Specifies the Team Server port number. This setting is not needed if you want to use the value specified in the GUI.

`tcm.server.accountLogin=true|false`
`tcm.server.username=[username]`
`tcm.server.password=[password]`
	Determines whether username and password are submitted to connect to Team Server. Usernames/passwords are not always needed; it depends on your team's setup.
	
	If the first setting is `true`, the second and third settings specify the username and password.
	
	Note that Team Server must have the username and password setting already enabled before these settings can be used.

Licensing Settings

Setting	Purpose
`cpptest.license.use_network=true|false`	Determines whether the current C++test installation retrieves its license from LicenseServer. This setting is not needed if you want to use the value specified in the GUI. Example: `cpptest.license.use_network=tru`
`cpptest.license.network.host=[host]`	Specifies the machine name or IP address of the machine running LicenseServer Configuration Manager. This setting is not needed if you want to use the value specified in the GUI. Example: `cpptest.license.network.host=10.9.1.63`
`cpptest.license.network.port=[port]`	Specifies the LicenseServer port number. This setting is not needed if you want to use the value specified in the GUI. Example: `cpptest.license.network.port=2222`
`cpptest.license.network.edition=[edition_name]`	Specifies the type of license that you want this C++test installation to retrieve from LicenseServer. This setting is not needed if you want to use the value specified in the GUI. `[edition_name]` can be `server_edition`. To use a custom edition, do not set anything after the "="; simply leaving the value empty. Example: `cpptest.license.network.edition=cpptest.license.network.edition=server_edition`
`cpptest.license.autoconf.timeout=[seconds]`	Specifies the maximum number of seconds C++test will wait for the license to be automatically configured from LicenseServer. Default is `10`.
`cpptest.license.local.expiration=[expiration]`	Specifies the local license that you want this C++test installation to use. This setting is not needed if you want to use the value specified in the GUI.
`cpptest.license.local.password=[password]`	Specifies the local password that you want this C++test installation to use. This setting is not needed if you want to use the value specified in the GUI.
`cpptest.wait.for.tokens.time=[time in minutes]`	Specifies the time that C++test will wait for a license if a license is not currently available. For example to make C++test wait 3 minutes for license tokens, use `cpptest.wait.for.tokens.time=3`.

Technical Support Settings

Setting	Purpose

`techsupport.auto_creation=true|false`	Determines whether archives are automatically prepared when testing problems occur.
`techsupport.send_email=true|false`	Determines whether prepared archives are emailed to Parasoft support. If you enable this, be sure to specify email settings from the GUI or with the options in Reporting Settings.
`techsupport.archive_location=[directory]`	Specifies where archives are stored.
`techsupport.verbose=true|false`	Determines whether verbose logs are included in the archive. Note that this option cannot be enabled if the logging system has custom configurations. • Verbose logs are stored in the `xtest.log` file within the user-home temporary location (on Windows, this is `<drive>:\Documents and Settings\<user>\Local Settings\Temp\parasoft\xtest`). • Verbose logging state is cross-session persistent (restored on application startup). • The log file is a rolling file: it won't grow over a certain size, and each time it achieves the maximum size, a backup will be created.
`techsupport.verbose.scontrol=true|false`	Determines whether verbose logs include output from source control commands. Note that the output could include fragments of your source code.
`techsupport.item.general=true|false`	Determines whether general application logs are included.
`techsupport.item.environment=true|false`	Determines whether environment variables, JVM system properties, platform details, additional properties (memory, other) are included in the archive.
`techsupport.advanced=true|false`	Specifies if advanced options will be sent.
`techsupport.advanced.options=[option]`	Specifies any advanced options that the support team asked you to enter.
`techsupport.dtp.engine=true|false`	Specifies if additional data generated during analysis will be sent.

Authorship/Scope Settings

Setting	Purpose

`authors.mappings.location=team|local|shared`	Specifies where the authorship mapping file is stored. This setting defaults to `team` unless `local` or `shared` is specified.
	
	If set to `local` (recommended), authorship mappings can be set directly in the local settings. See `authors.mapping` and `authors.user{n}` for details.
	
	If set to `shared`, you can store mappings in a local file using the `authors.mappings.file` option.
	
	The `team` and `shared` options are deprecated. Files specified by these options should be in the previouslyused format of:
	
	`#author to author`
	`user1=user3`
	`user2=user3`
	`#author to email`
	`user3=me@parasoft.com`

`authors.mapping{n}=[from_user, to_user]`	Specifies a specific author mapping for `authors.mappings.location=local`, as described above.
	
	For example:
	
	`authors.mappings.location=local`
	`authors.mapping1=baduser,gooduser`
	`authors.mapping2=brokenuser,fixeduser`
	`authors.mapping3=olduser,newuser`

`authors.user{n}=[username, email, full_name]`	Specifies a specific author name and email for `authors.mappings.location=local`.
	
	For example:
	
	`authors.user1=dan,dan@parasoft.com,Dan Stowe`
	`authors.user2=jim,jim@parasoft.com,Jim White`

`authors.mappings.file=[path]`	Specifies the location of a "shared" file as described in `authors.mappings.location` above. For example: `authors.mappings.file=/home/user/dev/temp/author_mapping1.txt`
`authors.ignore.case=true|false`	Determines whether author names are case sensitive. If true, David and david will be considered the same user. If false, David and david will be considered two separate users.
`scope.sourcecontrol=true|false`	Determines whether C++test computes code authorship based on a data from a supported source control system. This setting is not needed if you want to use the value specified in the GUI.
`scope.local=true|false`	Determines whether C++test computes code authorship based on the local user. This setting is not needed if you want to use the value specified in the GUI.
`scope.recommended.computation=first|random`	Determines how C++test selects the Recommended Tasks for each developer — it can choose n developer tasks at random (the default) or select the first n developer tasks reported (n is the maximum number of tasks that C++test is configured to show each developer per day)
`scope.xmlmap=true|false`	Specifies whether C++test computes task assignment based on XML files that define how you want tasks assigned for particular files or sets of files (these mappings can be specified in the GUI then saved in an XML file).
`scope.xmlmap.file=[file]`	Specifies the name of the XML file that defines how you want tasks assigned for particular files or sets of files.

Source Control Settings

Defining multiple repositories of the same type

Indexes (numbered from 1 to n) must be added to the prefix if you want to define more than one repository of the same type. For example:

```
scontrol.rep1.type=ccase
scontrol.rep1.ccase.vob=/vobs/myvob1

scontrol.rep2.type=ccase
scontrol.rep2.ccase.vob=/vobs/myvob2
```

If you are defining only one repository, you do not need to use an index. For example:

```
scontrol.rep.type=ccase

scontrol.rep.ccase.vob=/vobs/myvob1
```

AccuRev Repository Definition Properties

Property	Description
`scontrol.rep.type=accurev`	AccuRev repository type identifier.
`scontrol.rep.accurev.host=`	AccuRev server host.
`scontrol.rep.accurev.port=`	AccuRev server port. Default port is 1666.
`scontrol.rep.accurev.login=`	AccuRev user name.
`scontrol.rep.accurev.password=`	AccuRev password.

ClearCase Repository Definition Properties

Property	Description
`scontrol.ccase.exec=`	Path to external client executable (`cleartool`).
`scontrol.rep.type=ccase`	ClearCase repository type name.
`scontrol.rep.ccase.vob=`	Path inside VOB. ccase.vob value + File.separator must be the valid path to a ClearCase controlled directory.

CVS Repository Definition Properties

Property	Description
`scontrol.rep.type=cvs`	CVS repository type identifier.
`scontrol.rep.cvs.root=`	Full CVSROOT value.
`scontrol.rep.cvs.pass=`	Plain or encoded password. The encoded password should be the same as in the .cvspass file. For CVS use the value in `.cvspass` from within the user's home directory For CVSNT use the value store in the registry under `HKEY_CURRENT_USER\Software\Cvsnt\cvspass` When you are first logged in to the CVS repository from the command line using "`cvs login`", the password is saved in the registry. To retrieve it, go to the registry (using regedit), and look for the value under `HKEY_CURRENT_USERCVSNT>cvspass`. This should display your entire login name (`:pserver:exampleA@exampleB:/exampleC`) encrypted password value.
`scontrol.rep.cvs.useCustomSSHCredentials=`	Determines whether the cvs login and password should be used for EXT/SSH connections. Allowed values are `true` and `false`. It is disabled by default.
`scontrol.rep.cvs.ext.server`	If connecting to a CVS server in EXT mode, this specifies which CVS application to start on the server side. Has the same meaning as the CVS_SERVER variable `.cvs` is the default value.
`scontrol.rep.cvs.ssh.loginname=`	Specifies the login for SSH connections (if an external program can be used to provide the login).
`scontrol.rep.cvs.ssh.password=`	Specifies the password for SSH connection.
`scontrol.rep.cvs.ssh.keyfile=`	Specifies the private key file to establish an SSH connection with key authentication.
`scontrol.rep.cvs.ssh.passphrase=`	Specifies the passphrase for SSH connections with the key authentication mechanism.
`scontrol.rep.cvs.useShell=`	Enable an external program (CVS_RSH) to establish a connection to the CVS repository. Allowed values are `true` and `false`. It is disabled by default.
`scontrol.rep.cvs.ext.shell=`	Specifies the path to the executable to be used as the CVS_RSH program. Command line parameters should be specified in the `cvs.ext.params` property.
`scontrol.rep.cvs.ext.params=`	Specifies the parameters to be passed to an external program. The following casesensitive macro definitions can be used to expand values into command line parameters: • `{host}` repository host • `{port}` port • `{user}` cvs user • `{password}` cvs password • `{extuser}` parameter cvs.ssh.loginname • `{extpassword}` parameter cvs.ssh.password • `{keyfile}` parameter cvs.ssh.keyfile • `{passphrase}` parameter cvs.ssh.passphrase

Git Repository Definition Properties

Property	Description
`scontrol.rep.type=git`	Git repository type identifier.
`scontrol.git.exec=`	Path to Git executable. If not set, assumes git command is on the path.
`scontrol.rep.git.branch=`	The name of the branch that the source control module will use. This can be left blank and the currently checked out branch will be used.
`scontrol.rep.git.url=`	The remote repository URL (e.g., `git://hostname/repo.git`).
`scontrol.rep.git.workspace=`	The directory containing the local git repository.

Perforce Repository Definition Properties

Property	Description
`scontrol.perforce.exec=`	Path to external client executable (p4).
`scontrol.rep.type=perforce`	Perforce repository type identifier.
`scontrol.rep.perforce.host=`	Perforce server host.
`scontrol.rep.perforce.port=`	Perforce server port. Default port is 1666.
`scontrol.rep.perforce.login=`	Perforce user name.
`scontrol.rep.perforce.password=`	Password.
`scontrol.rep.perforce.client=`	The client workspace name, as specified in the P4CLIENT environment variable or its equivalents. The workspace's root dir should be configured for local path (so that files can be downloaded).

Serena Dimensions Repository Definition Properties

Linux Configuration Note

To use Serena Dimensions with C++test,Linux users should run in an environment prepared for using Serena programs, such as 'dmcli'

- `LD_LIBRARY_PATH` should contain the path to `<SERENA Install Dir>/libs`.
- `DM_HOME` should be specified.

Property	Description
`scontrol.rep.type=serena`	Serena Dimensions repository type identifier.
`scontrol.rep.serena.host=`	Serena Dimensions server host name.
`scontrol.rep.serena.dbname=`	Name of the database for the product you are working with.
`scontrol.rep.serena.dbconn=`	Connection string for that database.
`scontrol.rep.serena.login =`	Login name.
`scontrol.rep.serena.password`	Password.
`scontrol.rep.serena.mapping`	Maps workspace resources to Serena Dimension repository paths. • Example 1: If you use `scontrol.rep.serena.mapping_1=${project_loc\:MyProject};PRODUCT1\:WORKSET1;src\\MyProject`, then Project MyProject' will be mapped to the Serena workset `PRODUCT1:WORKSET1` and workset relative path: src\\MyProject • Example 2: If you use `scontrol.rep.serena.mapping_2=${workspace_loc};PRODUCT1\:WORKSET1`, then the complete workspace will be mapped to the Serena workset `PRODUCT1:WORKSET1`.

StarTeam Repository Definition Properties

Property	Description

`scontrol.rep.type=starteam`	StarTeam repository type identifier.
`scontrol.rep.starteam.host=`	StarTeam server host.
`sscontrol.rep.starteam.port=`	StarTeam server port. Default port is 49201.
`scontrol.rep.starteam.login=`	Login name.
`scontrol.rep.starteam.password=`	Password (not encoded).

`scontrol.rep.starteam.path=`	When working with large multi-project repositories, you can improve performance by specifying the project, view, or folder that you are currently working with.
	
	You can indicate either a simple Project name (all views will be scanned when searching for the repository path), a Project/View (only the given view will scanned) or Project/View/Folder (only the specified Star Team folder will be scanned).
	
	Examples:
	`scontrol.rep.starteam.path=proj1`
	
	`scontrol.rep.starteam.path=proj1/view1`
	
	`scontrol.rep.starteam.path=proj1/view1/folderA`
	
	`scontrol.rep.starteam.path=proj1/view1/folderA/folderB`

`scontrol.rep.starteam.workdir=`	If the `scontrol.rep.starteam.path` setting specifies a StarTeam view or folder, you can use this field to indicate 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).
	
	Examples:
	
	`scontrol.rep.starteam.workdir=c:\\storage\\dv`
	
	`scontrol.rep.starteam.workdir=/home/storage/dv`

Subversion Repository Definition Properties

Property	Description
`scontrol.rep.type=svn`	Subversion repository type identifier.
`scontrol.rep.svn.url=`	Subversion URL specifies protocol, server name, port and starting repository path (e.g., `svn://buildmachine.foobar.com/home/svn`).
`scontrol.rep.svn.login=`	Login name.
`scontrol.rep.svn.password =`	Password (not encoded).
`scontrol.svn.exec=`	Path to external client executable (`svn`).

CM Synergy Repository Definition Properties

Property	Description
`scontrol.rep.type=synergy`	Synergy/CM repository type identifier
`scontrol.rep.synergy.host=`	Computer on which synergy/cm engine runs. Local host is used when missing. *For Web mode, the host must be a valid Synergy Web URL with protocol and port (e.g., `http://synergy.server:8400`).*
`scontrol.rep.synergy.dbpath=`	Absolute synergy database path e.g `\\host\db\name` (backslash symbols '\' in `UNC/Windows` paths must be doubled).

`scontrol.rep.synergy.projspec=`	Synergy project spec which contains project name and its version e.g name-version.
`scontrol.rep.synergy.login=`	Synergy user name.
`scontrol.rep.synergy.password=`	Synergy password (not encoded).
`scontrol.rep.synergy.port=`	Synergy port.
`scontrol.rep.synergy.remote_client=`	(UNIX only) Specifies that you want to start ccm as a remote client. Default value is false. Optional. *This is not used for Web mode.*
`scontrol.rep.synergy.local_dbpath=`	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.*
`scontrol.synergy.exec=`	Path to external client executable (`ccm`)

Microsoft Visual Source Safe Repository Definition Properties

Property	Description
`scontrol.rep.type=vss`	Visual SourceSafe repository type identifier.
`scontrol.rep.vss.ssdir=`	Path of repository database (backslash symbols '\' in `UNC/Windows` paths must be doubled).
`scontrol.rep.vss.projpath=`	VSS project path.
`scontrol.rep.vss.login=`	VSS login.
`scontrol.rep.vss.password=`	VSS password.
`scontrol.vss.exec=`	Path to external client executable (`ss`).
`scontrol.vss.lookup=`	Determines whether a full VSS database search is performed to find associations between local paths and repository paths. `True` or `false`.

Important Notes

- The repository(n).vss.ssdir property should contain a UNC value even if the repository database resides locally.
- Be aware of VSS Naming Syntax, Conventions and Limitations. Any character can be used for names or labels, except the following:
  - Dollar sign ($)
  - At sign (@)
  - Angle brackets (< >), brackets ([ ]), braces ({ }), and parentheses (( ))
  - Colon (:) and semicolon (;)
  - Equal sign (=)
  - Caret sign (^)
  - Exclamation point (!)
  - Percent sign (%)
  - Question mark (?)
  - Comma (,)
  - Quotation mark (single or double) (' ")
- VSS 6.0 (build 8163), which is deployed with Visual Studio 6, does not work properly with projects whose names start with a dot (.) symbol. If such a project name is used, subprojects cannot be added.
- Do not use custom working directories for sub-projects (example: `Project $/SomeProject` has the working directory `C:\TEMP\VSS\SomeProject` and its subproject `$/SomeProject/SomeSubProject` has the working directory `D:\SomeSubProject`).

Settings for Creating BDF-Based Projects

Option	Description

`bdf.import.location=[WORKSPACE|BDF_LOC|<path>]`	You can specify an external location, or use the keyword `WORKSPACE`. If `WORKSPACE` is used, projects will be created in subdirectories within the workspace directory. If `BDF_LOC` is used and one project will be created, then it will be created in the exact location as the bdf file. If more then one project will be created, then the projects will be created in subdirectories within the bdf file location. Those subdirectories will be named with corresponding projects names. If an external path is specified, then the project will be created in the specified location. `WORKSPACE` is the default. For details on the available project creation options and their impacts, see Working with C++test Projects.
`bdf.import.pathvar.enabled=[true|false]`	Specifies if Path Variables should be used in linked folders that will be created in the new projects. The default is false.
`bdf.import.pathvar.name=<name>`	Specifies the name of the Path Variable (if Path Variables are used, per the bdf.import.pathvar.enabled property). The default Path Variable name is `DEVEL_ROOT_DIR`.
`bdf.import.pathvar.value=<path>`	Specifies the value of the Path Variable (if Path Variables are used, per the bdf.import.pathvar.enabled property). The default value is the most common root directory for all linked folders.
`bdf.import.compiler.family=<compiler_family>`	Specifies what compiler family will be used (for example, `vc_6_0, vc_7_0, vc_7_1, vc_8_0, gcc_2_9, gcc_3_2, gcc_3_3, gcc_3_4, ghs_4_0`). For a custom compiler, you need to use the custom compiler family identifier, which is the name of the directory containing gui.properties, c.psrc and cpp.psrc files). If this property is not specified, the default values will be used.
`bdf.import.c.compiler.exec=<exec>`	Specifies the executable of the C compiler that will be used in the created project.
`bdf.import.cpp.compiler.exec=<exec>`	Specifies the executable of the C++ compiler that will be used in the created project.
`bdf.import.linker.exec=<exec>`	Specifies the executable of the linker that will be used in the created project.
`bdf.import.project.<proj_name>=dir1;dir2;dir3`	Specifies the set of folders to link for the project `prj_name`. Folders should be specified as a value list of folder paths, separated with semicolons.

Settings for Importing Green Hills .gpj Projects

Setting	Purpose
`gpj.import.location=WORKSPACE|ORIG|<path>`	Specifies the location of the imported projects. If `WORKSPACE` is used, then the project will be created in workspace. If `ORIG` is used, then the project will be created in the `.gpj` project location. If an external path is specified, then the project will be created in the specified location. The default value is `WORKSPACE`.
`gpj.import.linked=true|false`	Specifies whether the `.gpj` project source folders are linked into the created Eclipse project. The default value is `true`.
`gpj.import.subdirs=true|false`	Applicable when `gpj.import.location=<path>` Specifies whether the project(s) are imported into subdirectories or directly into the specified location. If you want the project(s) imported into subdirectories created in the specified external location, use true. If you are importing only one project and you want it imported directly into the specified external location, use false. The default value is true (subfolders are created for each project imported into in external location).

`gpj.import.pathvar.enabled=true|false`	Specifies if path variables should be used when creating linked directories (if the above option is set to true). The default value is `false`.
`gpj.import.pathvar.name=<name>`	Specifies the path variable name. The default value will be used unless you specify a path variable name that points to a different location (for instance, `DEVEL_ROOT_DIR`). If a project with the specified name is already defined in the Eclipse workspace and it points to a different location than the value passed in the `gpj.import.pathvar.location` property, then Path Variable will not be used; full paths will be used instead. Also the default value of `gpjimport.pathvar.name` will always be `DEVEL_ROOT_DIR` if the `gpjimport.pathvar.name` property is not specified. If this property is specified with some <name>, then that <name> will be used as the Path Variable name. The default value is `DEVEL_ROOT_DIR`.
`gpj.import.pathvar.value=<path>`	Specifies the path variable value. By default, C++test calculates the common root for all linked folders.
`gpj.import.compiler.family=name`	Specifies the compiler family (compiler ID)
`gpj.import.c.compiler.exec=name`	Specifies the C compiler executable
`gpj.import.cpp.compiler.exec=name`	Specifies the C++ compiler executable
`gpj.import.linker.exec=name`	Specifies the linker executable

Settings for Importing IAR Embedded Workbench .ewp Projects

Setting	Purpose
ewp.import.location=WORKSPACE|EWP_LOC|<path>	Specifies the location of the imported projects. If `WORKSPACE` is used, then the project will be created in workspace. If `EWP_LOC` is used, then the project will be created in the .ewp project location. If an external path is specified, then the project will be created in the specified location. The default value is `WORKSPACE`.
ewp.import.config=<name>	Specifies which .ewp project configuration should be used. If the specified configuration cannot be found in the imported project, then the default configuration will be used. The configuration name can be passed in two ways: <project_name> - <configuration_name> or only <configuration_name>. If more then one project is imported, then only <configuration_name> should be entered. This prompts the wizard to search for that configuration in all projects. The default value is the default from .ewp.
ewp.import.linked=true|false	Specifies whether the .project source folders are linked to the created Eclipse project. Default: `true`.
ewp.import.subdirs=true|false	Specifies whether the project(s) are imported into subdirectories or directly into the specified location. Applicable when e`wp.import.location=<path>` is used. Set to `true` to import the project(s) into subdirectories created in the specified external location. Default. Set to `false` to import a single project directly into the specified external location.

ewp.import.pathvar.enabled=true|false	Set to `true` to use path variables when creating linked directories. The default value is `false`.
ewp.import.pathvar.name=name	Specifies the path variable name. The default name is `DEVEL_ROOT_DIR` and will be used unless a path variable name that points to a different location is specified.
ewp.import.pathvar.value=<path>	Specifies the path variable value. By default, C++test calculates the common root for all linked folders.

Settings for Importing Microsoft Visual Studio 6.0 .dsp Projects

Setting	Purpose
`dsp.import.location=WORKSPACE|DSP_LOC|<path>`	Specifies the location of the imported projects. If `WORKSPACE` is used, then the project will be created in workspace. If `DSP_LOC` is used, then the project will be created in the `.dsp` project location. If an external path is specified, then the project will be created in the specified location. The default value is `WORKSPACE`.
`dsp.import.linked=true|false`	Specifies whether the .dsp project source folders are linked into the created Eclipse project. The default value is `true`.
`dsp.import.subdirs=true|false`	Applicable when `dsp.import.location=<path>` Specifies whether the project(s) are imported into subdirectories or directly into the specified location. If you want the project(s) imported into subdirectories created in the specified external location, use true. If you are importing only one project and you want it imported directly into the specified external location, use false. The default value is true (subfolders are created for each project imported into in external location).
`dsp.import.pathvar.enabled=true|false`	Specifies if path variables should be used when creating linked directories (if the above option is set to true). The default value is `false`.
`dsp.import.pathvar.name=<name>`	Specifies the path variable name. The default value will be used unless you specify a path variable name that points to a different location (for instance, `DEVEL_ROOT_DIR`). If a project with the specified name is already defined in the Eclipse workspace and it points to a different location than the value passed in the `dsp.import.pathvar.location` property, then Path Variable will not be used; full paths will be used instead. Also the default value of `dsp.import.pathvar.name` will always be `DEVEL_ROOT_DIR` if the `dsp.import.pathvar.name` property is not specified. If this property is specified with some `<name>`, then that `<name>` will be used as the Path Variable name. The default value is `DEVEL_ROOT_DIR`.
`dsp.import.pathvar.location=<loc>`	Specifies what location the path variable points to. By default, the automatically-generated location will be used. This location is the common root location for all linked directories. If it is not possible to find a common location (for example because .dsp projects are on multiple drives) or the specified location cannot be used, then the path variable will not be used. Full paths will be used instead. For example, assume you have the following paths: path1: c:\a\b\c\proj1 path2: c:\a\b\proj2 The common root location would be `c:\a\b` The default value is automaticallygenerated.
`dsp.import.config=<name>`	Specifies which .dsp project configuration should be used. If the specified configuration cannot be found in the imported project, then the default configuration will be used. The configuration name can be passed in two ways: `<project_name> - <configuration_name>` or only `<configuration_name>`. If more then one project is imported, then only `<configuration_name>` should be entered. This prompts the wizard to search for that configuration in all projects. The default value is the default from `.dsp`.

For example, if the folder C:\temp\sources should be linked in an imported project and we have defined the path variable `DEVEL_ROOT_DIR` with the value C:\temp, then that folder will be linked as `DEVEL_ROOT_DIR/sources and the DEVEL_ROOT_DIR` path variable will be created in the workspace. If such a variable cannot be used (for example, because its value points to another folder not containing C:\temp\sources folder, it is already defined and has different value, or it has an invalid value), then C:\temp\sources folder will be linked using the full path C:\temp\sources.

Settings for Importing Keil uVision Projects

Setting	Purpose
`uv.import.location=WORKSPACE|ORIG|<path>`	Specifies the location of the imported projects. If `WORKSPACE` is used, then the project will be created in workspace. If `ORIG` is used, then the project will be created in the original project file location. If an external path is specified, then the project will be created in the specified location. The default value is `WORKSPACE`.
`uv.import.linked=true|false`	Specifies whether the `uVision` project source folders are linked into the created Eclipse project. The default value is `true`.
`uv.import.subdirs=true|false`	Applicable when `uv.import.location=<path>` Specifies whether the project(s) are imported into subdirectories or directly into the specified location. If you want the project(s) imported into subdirectories created in the specified external location, use `true`. If you are importing only one project and you want it imported directly into the specified external location, use `false`. The default value is `true` (subfolders are created for each project imported into in external location).
`uv.import.pathvar.enabled=true|false`	Specifies if path variables should be used when creating linked directories (if the above option is set to `true`). The default value is `false`.
`uv.import.pathvar.name=<name>`	Specifies the path variable name. The default value will be used unless you specify a path variable name that points to a different location (for instance, `DEVEL_ROOT_DIR`). If a project with the specified name is already defined in the Eclipse workspace and it points to a different location than the value passed in the `uv.import.pathvar.location` property, then Path Variable will not be used; full paths will be used instead. Also, the default value of uv.import.pathvar.name will always be `DEVEL_ROOT_DIR` if the `uv.import.pathvar.name` property is not specified. If this property is specified with a `<name>`, then that `<name>` will be used as the Path Variable name. The default value is `DEVEL_ROOT_DIR.`
`uv.import.pathvar.value=<path>`	Specifies the path variable value. By default, C++test calculates the common root for all linked folders.
`uv.import.config=<name>`	Specifies the name of the build configuration to use.

Settings for Importing Renesas High-performance Embedded Projects

Setting	Purpose

`hew.import.location=WORKSPACE|ORIG|<path>`	Specifies the location of the imported projects. If `WORKSPACE` is used, then the project will be created in workspace. If `ORIG` is used, then the project will be created in the original project file location. If an external path is specified, then the project will be created in the specified location. The default value is `WORKSPACE`.
`hew.import.linked=true|false`	Specifies whether the `HEW` project source folders are linked into the created Eclipse project. The default value is `true`.
`hew.import.subdirs=true|false`	Applicable when `hew.import.location=<path>` Specifies whether the project(s) are imported into subdirectories or directly into the specified location. If you want the project(s) imported into subdirectories created in the specified external location, use `true`. If you are importing only one project and you want it imported directly into the specified external location, use `false`. The default value is `true` (subfolders are created for each project imported into in external location).
`hew.import.pathvar.enabled=true|false`	Specifies if path variables should be used when creating linked directories (if the above option is set to true). The default value is `false`.
`hew.import.pathvar.name=<name>`	Specifies the path variable name. The default value will be used unless you specify a path variable name that points to a different location (for instance, `DEVEL_ROOT_DIR`). If a project with the specified name is already defined in the Eclipse workspace and it points to a different location than the value passed in the `hew.import.pathvar.location` property, then Path Variable will not be used; full paths will be used instead. Also the default value of `hew.import.pathvar.name` will always be `DEVEL_ROOT_DIR` if the `hew.import.pathvar.name` property is not specified. If this property is specified with a `<name>`, then that `<name>` will be used as the Path Variable name. The default value is `DEVEL_ROOT_DIR`.
`hew.import.pathvar.value=<path>`	Specifies the path variable value. By default, C++test calculates the common root for all linked folders.
`hew.import.config=<name>`	Specifies the name of the build configuration to use.

Miscellaneous Settings

Setting	Purpose

`report.rules=[url_path_to_rules_directory]`	Specifies the directory for rules html files (generated by clicking the `Printable Docs` button in the Test Configuration's Static Analysis tab).
	
	For example:
	
	`report.rules=file:///C:/Temp/Burt/parasoft/xtest/gendoc/`
	
	`report.rules=../gendoc/`
	
	The default setting is none.

`tasks.clear=true|false`	Clears existing tasks upon startup in cli mode. This prevents excessive time being spent "loading existing results." The default is `true`.

`console.verbosity.level=low|normal|high`	Specifies the verbosity level for the Console view. Available settings are: `low`: Configures the Console view to show errors and basic information about the current step's name and status (done, failed, up-to-date). `normal`: Also shows command lines and issues reported during test and analysis. `high`: Also shows warnings.
`cpptest.custom.rules.dir=[directory]`	Indicates where user-defined rules are saved.
`cpptest.custom.configs.dir=[directory]`	Indicates where user-defined Test Configurations are saved.
`custom.compilers.dir=[directory]`	Overrides the custom compiler directory settings (found in **Parasoft> Configurations> Custom compilers**) and uses the defined directory to search for custom compilers
`parallel.mode=Manual|Auto|Disabled`	Determines which of the following modes is active: • **Auto**: Allows Parasoft Test to control parallel processing settings. • **Manual**: Allows you to manually configure parallel processing settings to suit your specific needs. • **Disabled**: Configures Parasoft Test to use only one of the available CPUs. For more details on this and other parallel processing options, see Configuring Parallel Processing.
`parallel.max_threads=<number>`	Specifies the maximum number of parallel threads that can be executed simultaneously. The actual number of parallel threads is determined by the number of CPUs, available memory, and license settings.
`parallel.free_memory_limit=<percentage>`	Specifies the amount of memory that should be kept free in low memory conditions (expressed as a percentage of the total memory available for the application). This is used to ensure that free memory is available for other processes.
`parallel.no_memory_limit=true|false`	Indicates that you do not want to place any restrictions (beyond existing system limitations) on the memory available to C++test

Here is one sample options file named `local.properties`:

```
# Team Server settings: (these may be redundant with settings already specified in Team Preferences of the
installed version, so may not be needed).
tcm.server.enabled=true
tcm.server.name=<team_server.company.com>

# Report settings
report.developer_errors=true
report.developer_reports=true
report.format=html
session.tag=<project name>

# Mail settings:
report.mail.enabled=true
report.mail.cc=<manager1@mailserver.com1;manager2@mailserver.com1>
report.mail.server=mail.company.com
report.mail.domain=company.com
report.mail.subject=<Static Analysis results on Project X>
report.mail.attachments=true
```

Here is another sample:

```
# Team Server settings
tcm.server.enabled=true
tcm.server.name=teamserver.mycompany.com
tcm.server.port=18888
tcm.server.accountLogin=true
tcm.server.username=tcm_user
tcm.server.password=tcm_pass

# Parasoft DTP settings
dtp.server=dtp.mycompany.com
dtp.port=32323

# Mail settings
report.mail.enabled=true
report.mail.server=mail.mycompany.com
report.mail.domain=mycompany.com
report.mail.cc=project_manager
report.mail.subject=Coding Standards
concerto.log_as_nightly=true
```

Using Variables in Local Settings (Options) Files

The following variables can be used in reports, e-mail, Parasoft DTP, Team Server, and license settings. Note that session tag value can't contain any ':' characters.

env_var

example: ${env_var:HOME}

Outputs the value of the environmental variable specified after the colon.

project_name

example: ${project_name}

Outputs the name of the tested project. If more than one project is provided as an input, it first outputs the tested project name, then "..."

workspace_name

example: ${workspace_name}

Outputs an empty string.

config_name

$ example: ${config_name}

Outputs the name of executed Test Configuration; applies only to Reports and Email settings.

analysis_type

$ example: ${analysis_type}

Outputs a comma separated list of enabled analysis types (for example: Static, Generation, Execution); applies only to Reports and Email settings.

tool_name

$ example: ${tool_name}

Outputs the tool name (for example: C++test).

Example localsettings file

```
# REPORTS
#Determines whether reports are emailed to developers and to the additional recipients specified with the cc
setting.
#Remember that if the team is using CVS for source control and each developer's email address matches his or
her CVS username + the mail domain, each developer that worked on project code will automatically be sent a
report that contains only the errors/results related to his or her work.
report.mail.enabled=true
#Exclude developers emails (true/false)
```

```
report.mail.exclude.developers=false
# Append developers errors to manager emails (true/false) report.developer_errors=true
# Send reports to developers (true|false) report.developer_reports=true
# Append suppressed messages (true|false) report.suppressed_msgs=false#Determines where to mail complete test
reports.
#This setting is typically used to send reports to managers or architects.
#It can also be used to send reports to developers if developer reports
#are not sent automatically (for example, because the team is not using CVS).
report.mail.cc=manager@domain.com; ${env_var:USERNAME} @domain.com
# mail target for unknown developer errors report.mail.unknown=manager@domain.com
#Specifies the mail server used to send reports.
report.mail.server=mail_server.domain.com
#Specifies the mail domain used to send reports.
report.mail.domain=domain.com
#Specify mali from report.mail.from=nightly
#Specifies any email addresses you do not want to receive reports.
#This setting is used to prevent from automatically sending reports to someone that worked on the code, but
should not be receiving reports. This setting is only applicable if the team is using CVS for source control
and developer reports are being sent automatically.
report.mail.exclude=developer1;developer2
# Specifies the subject line of the emails sent.
report.mail.subject= ${tool_name} Report - ${config_name}
# Report test params include (true|false)
report.test_params=true
# Team Server
#Determines whether the current installation is connected to the Team Server.
tcm.server.enabled=true
#Specifies the machine name or IP address of the machine running Team Server.
tcm.server.name=team_server.domain.com
#Specifies the Team Server port number.
tcm.server.port=18888
tcm.server.accountLogin=true
tcm.server.username=user
tcm.server.password=password
session.tag= ${config_name}
# Parasoft Project Center
#Determines if the current installation is connected to Parasoft Project Center.
concerto.reporting=true
#Specifies the host name of the Parasoft DTP server.
dtp.server=grs_server.domain.com
# Specifies the port number of the Parasoft Project Center report collector.
concerto.data.port=32323
# Specifies user-defined attributes for Parasoft Project Center.
#Use the format key1:value1; key2:value2
#Attributes help you mark results in ways that are meaningful to your organization.
#They also determine how results are grouped in Parasoft Project Center and how you can filter results in
Parasoft Project Center.
#For example, you might want to label results by project name and/or by project component name.
#Each attribute contains two components: a general attribute category name
#and a specific identification value. For example, assume your organization wants to classify results by
project.
#You might then use the attribute project:projname1. For the next project, you could use a different
#local settings file that specified an attribute such as project:projname2.
concerto.user_defined_attributes=Type:Nightly;Project:Project1
# Determines whether the results sent to Parasoft Project Center are marked as being from a nightly build.
concerto.log_as_nightly=true
# SCOPE
#code authorship based on CVS
scope.sourcecontrol=true
#code authorship based on author tag
scope.author=false
#code authorship based on local user
scope.local=false
# LICENSE
#override license settings
#cpptest.license.autoconf.timeout=40
cpptest.license.use_network=true
cpptest.license.network.host=license_server.domain.com
cpptest.license.network.port=2222
cpptest.license.network.edition=server_edition
# SOURCE CONTROL
```

```
scontrol.rep1.type=cvs
scontrol.rep1.cvs.root=:pserver:developer@cvs_server.domain.com:/home/cvs/
scontrol.rep1.cvs.pass=mypassword
```

Using the cli with an Eclipse-Based Builder

The `-buildscript %SCRIPT_FILE%` option executes the specified Eclipse build script prior to testing.

In some cases, this may bring up the Eclipse UI. It depends on the Eclipse component capabilities and 3rd party Eclipse source control plugins. Note that if the UI is not opened and the source control is not fully configured, then the 3rd party Eclipse source control plugins may fail in headless mode. They may fail silently or throw various exceptions while trying to access a UI that is not available. To prevent this, have source control fully configured and ensure that it does not need to ask the user for any additional information (username, passwords, etc.)

The following scripting language can be used to define the script...

Syntax

Commands are entered one per line. Whitespace at the beginning or end of the line is trimmed. Any blank line is ignored. Everything following a # comment symbol in a line is ignored. Commands consist of a command name and one or more arguments.

Substrings of the form `$(key)` are recursively expanded as macros in arguments.

Commands

Command	Description
`var <name> <value>`	Defines a variable that will be used in macro expansion.
`co </path/to/file.psf>`	Checks out projects specified in an Eclipse team project set exported file. Relative paths are resolved relative to the current file (important for included files).
`up </path/to/file.psf>`	Exactly like co, but if a project is already in the workspace, it will update it rather than checking it out from scratch. Tries to do this "headlessly" but may open a dialog, suspending the checkout until user interaction is given via the UI to resolve the conflict, if local changes are detected. NOTE: Updating is only supported with CVS and SVN repositories. If the team project set refers to another kind of repository, only checkout is supported at this time.
`upb </path/to/file.psf>`	Exactly like co, but if a project is already in the workspace, it will update it rather than checking it out from scratch. Same advantages and limitations as "up". Builds all projects in the PSF file after checkout and/or update is complete.
`rep </path/to/file.psf>`	Exactly like up, but if a project already in the workspace is being updated, it uses a replace operation rather than an update operation. This should overwrite local changes silently. NOTE: Updating/replacing is only supported with CVS and SVN repositories. If the team project set refers to another kind of repository, only checkout is supported at this time.
`repb </path/to/file.psf>`	Exactly like upb, but if a project already in the workspace is being updated, it uses a replace operation rather than an update operation. This should overwrite local changes silently. Builds all projects in the PSF file after checkout and/or update is complete.
`build [project1 [project2[...]]]`	Builds the projects whose names are given in the arguments. If no argument is given, builds all projects in the workspace. If the first argument is the string "-", builds all projects in the workspace except the ones listed.
`include </path/to/script>`	The specified script file will be run. Relative paths are resolved relative to the current file (important for recursive include commands).
`ant </path/to/build.xml> [target1 [target2 [...]]]`	The specified targets (or the default target, if no targets are specified) in the specified ant build file will be run. Relative paths are resolved relative to the current file. Refresh of entire workspace is done after the ant build file is run.
`refresh`	Performs an Eclipse "refresh" of the entire workspace.

Macros

Strings of the form `$(key)` in command arguments are expanded. The values used can be from previous var commands or from System properties. System properties can be predefined by Java (e.g. `user.home`) or passed into the build by running Eclipse with the `-vmargs -Dkey=value` parameter.