Testing from the Command Line Interface

This topic explains how to run a test from the C/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/C++test Automation Edition).

Extended Command Line Mode vs. Desktop Command Line Mode

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

1. **Extended Command Line Mode** is provided in Automation Edition 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 Automation Edition, you also need to install and configure Parasoft Team Server.
- We strongly recommend that you configure C/C++test preferences 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/C++test, it must be added to a Visual Studio 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/C++test Professional 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 Automation 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/C++test CLI can be invoked on a full Visual Studio solution, or one or more projects or source files that are part of a solution. As part of the CLI execution, C/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 testsin a given solution.

- 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/C++test are controlled via Test Configurations and Preferences.

Test Configurations can be sourced from the built in set, or created using C/C++test interactive mode in the GUI. We suggest using the built-in configurations as starting templates for customer-specific configurations.

Preferences can be configured from the C/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/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/C++test preferences are accessed through the **Parasoft> Preferences** menu. Start by configuring the following preferences:

- **License**: Specify the license or License Sever settings.
- **DTP:** Specify your DTP server settings.
- (Optional) **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 C/C++test, 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.

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, your command line may resemble the following:

- `cpptestcli -solution "c:\MySolution" -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:
    - **-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/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/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:

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.

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.
- `-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 violations or setup problems are reported (see Command Line Exit Codes for details about exit codes returned if the process fails).
- `-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.

- `-disablescm` - Disconnects source control information from the solution and project.
  Disconnecting source control information from the solution and projects is useful, for example, to keep user input requests from reaching the source control plugin while running tests in cli mode. After running tests, source control information is restored. Disabling and reenabling source control is performed by modifying .sln solution file and .vcproj project files.
  To use this option, ensure that those files are writable (or run C++test from a user account with rights to make those files writable).
- `-solutionConfig %SOLUTION_CONFIG_NAME% -` Specifies the solution configuration to use for building the solution and for analysis. `Debug` and `Release` are common names. If the switch is omitted, then the active configuration is used. Specifying the solution configuration is strongly recommended because the active configuration may change unexpectedly.
- `-targetPlatform %TARGET_PLATFORM_NAME% -` Specifies the solution target platform to use for building the solution and for analysis. Any CPU and x86 are common names. If omitted, the active configuration is used. Specifying the target platform is strongly recommended because the active configuration may change unexpectedly.

- `-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. Must be used along with `-solution`.
- -list-configs - Prints a list of valid Test Configuration values. Must be used along with `-solution`.
- `-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"`

- `-useenv` - Causes the IDE to use PATH, INCLUDE, and LIB environment variables for Visual C++ compilation rather than the settings specified in the VC++ Directories section of the Projects options (in the Options dialog box).
- `-clearcmc` - Cleans Visual Studio's Component Model Cache

Options for Testing Projects Available in the Visual Studio IDE

Option	Purpose	Notes
`-solution %SOLUTION_FILE%-solution %SOLUTION_FILE%`Specifies the location of the solution file to use.	Specifies the location of the solution file to use.Specifies the location of the solution file to use.	N/AN/ASpecifies the location of the solution file to use.

`-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.
		
		For example, to test the `ATM.cxx` file in the C++test ATM example, you could use -resource "ATM/Source Files/ATM.cxx" (without the solution name)
		
		or
		
		`-resource "Examples/ATM/Source Files/ATM.cxx"` (with the solution name)
		
		Other Examples:
		
		`-resource "MySolution/MyProject"`
		
		`-resource "MySolution/MyProject/Source Files"`
		
		`-resource "MySolution/MyProject/SourceFiles/MyClass.cpp"`
		
		`-resource "c:\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 allows you to:

- 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.

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.

See Configuring Localsettings for information about localsettings files and the list of available settings.