Command Line Options

This page lists the command line options for `cpptestcli`.

Configuring the Test Configuration

`-config <config_url>`

This option allows you to specify the test configuration that will be used for analysis. The option must be followed by the name of a built-in, user-defined, or DTP-hosted test configuration. Examples:

- `cpptestcli -config "builtin://MISRA C"`
- `cpptestcli -config "user://My_Config"`
- `cpptestcli -config "dtp://New_Rules"`

`-listconfigs`

This option prints a list of available test configurations and can be used to obtain a valid test configuration name you can pass with the `-config` option.

See Configuring Test Configurations for details.

Configuring the Compiler

`-compiler <name|path>`

This option specifies compiler configuration name to be used for code analysis and instrumentation. Compiler configuration name needs to be one of the supported compilers names. See Supported Compilers or use the `-list-compilers` option.

`-list-compilers`

This option prints a list of supported compilers and can be used to obtain a vaild compiler ID you can pass with the `-compiler` option.

`-detect-compiler`

This option detects the compilers based on the compiler command (the executable name) you specify with or without the path). You can provide a list of compiler commands with a semicolon-separated list and use this option to obtain a valid compiler ID to pass with the `-compiler` option. Example:

- `-detect-compiler gcc`

Configuring the Input Scope

-trace <build command>

This option specifies the build command that starts the build process to compile the files to be tested. The information about the build process and the input scope data collected during the build are stored in the `cpptest.bdf` file. C/C++test-specific options must be specified before `-trace`, because all values following `-trace` will be interpreted as a build command. Example:

- `cpptestcli -config "builtin://Recommended Rules" -compiler gcc_3_4 -trace make clean all`

-input <build data file|project file>

This option specifies the path to an existing build data file (.bdf) or another project definition file to collect the input scope.

Build data file: You can create a build data file with the -trace option or by using the standalone `cpptestscan` or `cpptesttrace` utilities.

Project file: Currently, Microsoft Visual Studio projects and solutions (.vcproj, .vcxproj, .sln) are supported. Additionally, when specifying Microsoft Visual Studio project or solution you can provide configuration and platform to collect data for <PROJECT_FILE>@<CONFIGURATION>|<PLATFORM>.

Examples:

- `-input cpptest.bdf`
- `-input MyProject.vcproj`
- `-input MySolution.sln`
- `-input "MySolution.sln@Release|Win32"`

-- <compile command

A switch to collect the input scope form a compile command. Ensure the compile command is complete and includes your compiler executable, compiler flags, and source files.

C/C++test-specific options must be specified before `--` switch, because all values after `--` switch will be interpreted as a compiler command. Example:

- `cpptestcli -config "builtin://Recommended Rules" -compiler gcc_3_4 -- gcc -I include Bank.cxx`

-module [<module name>=]<module root directory>

This option specifies the module root directory and associates it with a module. If the module name is not specified, C/C++test uses the directory name as the module name.

This option allows you define modular structure for C/C++test files included in the .bdf file:

- a source file located in the specified module root directory (or its subdirectory) belongs to the associated module
- a header file located in the specified module root directory (or its subdirecotyr) beliongs to the associated module
- a header file included in a tested source file is tested only if it belongs to the same module as the source file

You can specify this option more than once to define multiple modules - for example, files included in one .bdf file can be divided into different modules.

This option is not supported for Microsoft Visual Studio projects / solutions (.vcproj, .vcxproj, .sln).

Examples:

- `-module MyProject=/home/project/src`
- `-module ../Module1`
- `-module .`

-resource <path|file name>

This option narrows down the input scope. You can specify one of the following:

- a path to a file – to test a selected file
- a path to a directory – to test the files included in the selected directory
- a file name - to only test the files that match the specified name
- a path to the `.lst` file that lists the resources to be analyzed (ech line in the file will be treated as a single entry)

Examples:

- `-resource /home/cpptest/examples/ATM/ATM.cxx`
- `-resource /home/cpptest/examples/ATM`
- `-resource ATM.cxx`
- `-resource c:/resource.lst`

-include <absolute path> and -exclude <absolute path>

These options include or exclude file(s) that match the specified pattern into/from instrumentation scope. The options can be specified multiple times. Final filtering is determined only after all include/exclude entries have been specified in the order of their specification.

Provide an absolute path to a file (you can use * as a wildcard). Examples:

- `-include /home/project/src/ATM.cxx`
- `-exclude /home/project/src/*.cxx`
- `-exclude /home/project/MyExcludes.lst`

Alternatively, you can specify a list of patterns in a `.lst` file and pass the file with the `-include` or `-exclude` switch. Each line in the file will be treated as a single entry. See Configuring the Test Scope for details.

Reporting

-report <path>

This option specifies the path to the directory where the report will be created.

-publish

This option sends the results to DTP. See Sending Results and Publishing Source Code to DTP.

Customizing Configuration

-settings <path>, -localsettings <path>, and -ls <path>

This option specifies the path to a custom .properties file that includes customized settings in the following format: `key=value` (for example, `report.format=pdf`). You can use this option multiple times to specify several .properties files. Entries with the same key will be overwritten. Example:

- `-settings Project1Config.properties`

-property <key>=<value>

This option allows you to configures a single setting directly in the command line. Use the following format: `key=value`.

You can use this option multiple times to configure several settings on the same command line. Earlier entries with the same key will be overwritten. Examples:

- `-property session.tag=sa_linux -property report.dtp.publish=true -property techsupport.create.on.exit=true`

-showsettings

This option prints the current settings and customizations.

-psrc <path|name>

This option specifies an advanced configuration file provided by Parasoft Support. Example:

- `-psrc advanced_options.psrc`

See Configuration Settings for the list of settings you can configure.

Other Options

-machineId

This option prints your machine ID.

-encodepass <your password>

This option prints an encoded password that can be used in the .properties configuration file.

-workspace <path>

This option specifies the workspace directory to be used during code analysis and instrumentation. The workspace location is used to store C/C++test data files (such as `cpptest.bdf`) and incremental data directory (`.cpptest`). If not specified, C/C++test uses your working directory for storage. Example:

- `-workspace /home/qa/workspace_for_project1`

You can run only one instance of C/C++test per workspace.

-showdetails

This option increases console verbosity to display progress details.

-fail

This option fails the command by returning a non-zero exit code if some findings are reported.

-version

This option prints information about the version of the Parasoft tool you are using.

-help

This option prints the command line help.