Running Static Analysis 1

In this section:

Executable

Verify that the executable (`cpptestcli`) is on $PATH. See Installation.

Incremental Static Analysis

C/C++test is optimized to minimize the impact on your development process. By reusing information collected in previous analysis runs, it can reduce the analysis time in subsequent runs. The information is stored in the .cpptest folder in your working directory (see the -workspace option).

To ensure optimal performance, avoid removing the .cpptest folder or deleting the contents of the folder.

Prerequisites

Compiler

C/C++test must be configured for use with specific C and C++ compilers and versions before you can analyze code. The configuration should reflect the original compiler and version used for building the code under test. The original compiler executable must be available on $PATH (unless it is specified with a full path).

Use the `-compiler` switch to specify the compiler configuration identifier:

```
cpptestcli -config "builtin://Recommended Rules" -compiler gcc_4_5 -input cpptest.bdf
```

Alternatively, you can configure the `cpptest.compiler.family` property in a custom configuration file:

```
cpptest.compiler.family=gcc_4_5
```

If you are using a single compiler and version for all testing, the compiler identifier can be specified in the `cpptestcli.properties` global configuration file in either the [INSTALL_DIR] or [USER_HOME] directory; see Configuration Overview.

Compiler Discovery

Perform one of the following actions to find the configuration for your compiler:

- Use the `-detect-compiler` switch to the auto-detect configuration:

  ```
  cpptestcli -detect-compiler gcc
  ```

- Use the `-list-compilers` switch to find the configuration in the list of all supported compilers:

  ```
  cpptestcli -list-compilers
  ```

See Configuring the Compiler and Supported Compilers for additional information.

About Usage Examples

The following instructions assume that:

- GNU GCC 3.4 compiler is being used (configuration identifier: gcc_3_4)

- The prerequisites discussed above have been met.
- Users are running commands in the [INSTALL_DIR]/examples/ATM directory.

Analyzing a Single File

Run an analysis and specify the original compiler command with the `--` switch (separator).

ⓘ All values after `--` switch will be interpreted as a compiler command, so options specific to C/C++test must be specified before `--` switch.

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

C/C++test will analyze the Bank.cxx file using the original compiler executable and compiler options and report detected violations to the output console.

Only the specified source files will be analyzed. Header files included by the source files will be excluded from analysis. To broaden the scope of files tested, including header files, see Configuring the Test Scope.

You can also use the `-fail` option to generate a non-zero exit code when the analysis reports static analysis findings (see Command Line Exit Codes).

Analyzing a Makefile-based Project

Run code analysis and specify the original **build command** with the `-trace` switch.

ⓘ All values after -trace will be interpreted as a build command, so options specific to C/C++test must be specified before `-trace`.

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

C/C++test will perform the following tasks:

1. Run the original build (make clean all)
2. Detect which files to test
3. Run the analysis for these files
4. Report results to the output console
5. Store all build information in the cpptest.bdffile for future runs (see About Build Data Files for additional information about build data files).

C/C++test will detect a source file for testing only if that file was compiled when running the build command. Only source files from Makefile will be analyzed. Header files included by the source files will be excluded from analysis. To broaden the scope of files tested, including header files, see Configuring the Test Scope.

About Build Data Files

You can create a build data file (.bdf), which stores information such as the working directory, command line options for the compilation,and link processes of the original build, so that C/C++test can analyze a project without having to rebuild it. The following example is a fragment from a build data file:

```
working_dir=/home/place/project/hypnos/pscom
project_name=pscom
arg=g++
arg=-c
arg=src/io/Path.cc
arg=-Iinclude
arg=-I.
arg=-o
arg=/home/place/project/hypnos/product/pscom/shared/io/Path.o
```

You can use the `-trace` switch to create a .bdf or use the standalone `cpptestscan` or `cpptesttrace` utility located in the [INSTALL_DIR]/bin directory.

Using `cpptestscan` and `cpptesttrace` Utilities

The `cpptestscan` utility is used as a wrapper for the compiler and/or linker during the normal build. To use `cpptestscan` with an existing build, prefix the compiler/linker executable with `cpptestscan` when building the code base. This can be done in two ways:

- Modify the build command line to use `cpptestscan` as the wrapper for the compiler/linker executables
- If you aren't able to override the compiler variable on the command line, embed`cpptestscan` in the actual make file or build script.

To use `cpptesttrace` with an existing build, prefix the entire build command with `cpptesttrace` when building the code base. `cpptesttrace` will trace the compiler and linker processes executed during the build and store them in the build data file.

In both cases, you must specify the full path to either utility in your PATH environment variable. Additional options for `cpptestscan` and `cpptesttrace` are summarized in the following table. Options can be set directly for the `cpptestscan` command or via environment variables. Most options can be applied to `cpptestscan` or `cpptesttrace` by changing the prefix in the command line.

Basic cpptestscan usage:

- Windows: `cpptestscan.exe [options] [compile/link command]`
- Linux and Solaris: `cpptestscan [options] [compile/link command]`

Basic cpptesttrace usage:

- Windows: `cpptesttrace.exe [options] [build command]`
- Linux, Solaris: `cpptesttrace [options] [build command]`

Option	Environment Variable	Description	Default

`--cpptestscanOutput-File=<OUTPUT_FILE>`
 	
`--cpptesttraceOutput-File=<OUTPUT_FILE>`
	CPPTEST_SCAN_OUTPUT_FILE	Defines file to append build information to.	cpptestscan.bdf

`--cpptestscanProject-Name=<PROJECT_NAME>`
 	
`--cpptesttraceProject-Name=<PROJECT_NAME>`
	CPPTEST_SCAN_PROJECT_NAME	Defines a suggested name of the C++test project.	name of the current working directory

`--cpptestscanRun-OrigCmd=[yes|no]`
 	
`--cpptesttraceRun-OrigCmd=[yes|no]`
	CPPTEST_SCAN_RUN_ORIG_CMD	If set to "yes",original command line will be executed.	yes

`--cpptestscanQuoteCmdLineMode=[all|sq|none]`
 	
`--cpptesttraceQuoteCmdLineMode=[all|sq|none]`
	CPPTEST_SCAN_QUOTE_CMD_LINE_MODE	Determines the way C++test quotes parameters when preparing cmd line to run.
		
		`all`: all params will be quoted
		
		`none`: no params will be quoted
		
		`sq`: only params with space or quote character will be quoted
		
		cpptestscanQuoteCmdLineMode is not supported on Linux
			all

`--cpptestscanCmd-LinePrefix=<PREFIX>`
`--cpptesttraceCmd-LinePrefix=<PREFIX>`
	CPPTEST_SCAN_CMD_LINE_PREFIX	If non-empty and running original executable is turned on, the specified command will be prefixed to the original command line.	[empty]

`--cpptestscanEnvInOutput=[yes|no]`
 	
`--cpptesttraceEnvInOutput=[yes|no]`
	CPPTEST_SCAN_ENV_IN_OUTPUT	Enabling dumps the selected environment variables and the command line arguments that outputs the file. For advanced settings use --cpptestscanEnvFile and --cpptestscanEnvars options	no

`--cpptestscanEnv-File=<ENV_FILE>`
 	
`--cpptesttraceEnv-File=<ENV_FILE>`
	CPPTEST_SCAN_ENV_FILE	If enabled, the specified file keeps common environment variables for all build commands; the main output file will only keep differences. Use this option to reduce the size of the main output file. Use this option with --cpptestscanEnvInOutput enabled	[empty]

Re-analyzing a Project without Re-building

Run code analysis and specify the existing build data file with the `-input` switch:

```
cpptestcli -config "builtin://Recommended Rules" -compiler gcc_3_4 -input cpptest.bdf
```

C/C++test will perform the following tasks:

1. Read the information about which files to test from the existingcpptest.bdffile
2. Run the analysis for these files
3. Report results to the output console
4. The original build will not be executed

Multiple build data files can be specified using multiple `-input` switches:

```
cpptestcli -config "builtin://Recommended Rules" -compiler gcc_3_4 -input project1.bdf -input project2.bdf
```

Only the source files defined in the build data file will be analyzed. Header files included by the source files will be excluded from analysis. To broaden the scope of files tested, including header files, see Configuring the Test Scope.

Testing a Microsoft Visual Studio Project or Solution

C/C++test can read Visual Studio project and solution files and analyze all source and included header files from the project or solution. Use the `-input` switch to specify a Visual Studio project or solution file:

```
cpptestcli -config "builtin://Recommended Rules" -input MyProject.vcproj
```

You can specify the build configuration and the platform you want to use during analysis of your project or solution. Append the configuration and platform names to the solution or project file name. Your command may resemble the following:

```
cpptestcli -config "builtin://Recommended Rules" -input MyProject.vcproj@Debug|x64
```

Alternatively, you can use the following properties to specify the build configuration and the platform you want to use during analysis of all Visual Studio solutions and projects:

```
cpptest.input.msvc.config=Debug
cpptest.input.msvc.platform=x64
```

For all Microsoft Visual Studio settings, see Visual Studio Settings.

Ensure that the correct version of Microsoft Visual C++ compiler is available on $PATH before running analysis. Microsoft Visual Studio 6 is not supported.

Specifying C/C++test Data Location

Exclusive access to the `.cpptest` directory is required. The directory is created in the current working directory by default, which is where some of the run-specific data is stored. As a result, only one instance of C/C++test can run in a directory at a time. You can use the `-workspace` switch to change the location of the .cpptest directory.

```
-workspace <WORKSPACE_LOCATION>
```