.Configuring+the+Test+Scope+v10.4.0

In this section:

Overview

The input scope defines the C and C++ source files to test with C/C++test. The input scope also provides the full set of information about compiler options and environment, so C/C++test can re-create the original build environment to provide accurate test results. See Running Static Analysis for information about defining compilers.

Analyzing a Single File

See Running Static Analysis for instructions.

Analyzing a Makefile-based Project

See Running Static Analysis for instructions.

Analyzing Code Using Existing Build Data

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. See the following sections for additional information:

- Description of the concept of the .bdf and how to create it, see Running Static Analysis.
- Description of the steps for using the .bdf for analysis, see Running Static Analysis.
- Description of how to broaden the scope of files tested, including header files, see Defining Source File Structures (Modules).

Defining Source File Structures (Modules)

C/C++test treats the input scope as a set of unrelated source files. Defining modules allows you to introduce a well-defined source file structure and add additional files, such as header files, into the Input Scope.

Modules are defined by specifying its name and the root directory. All tested files located in the root directory or its sub-directories will belong to the module. All header files located in the root directory or its sub-directories that are included by the tested source files will also belong to the module and be analyzed with the source files.

For all files from the module, a "module-relative path" will be available. A project-relative path is computed as a relative path from the module root to the actual file location. In most cases, module-relative paths are independent of machines, so the test results can be easily shared across different machines.

Example of Module Structure

The first block of code describes a simple directory/file structure. In the second block of code, the relationships between the files and module root directory are described, as well as which files will be analyzed:

`/home/devel_1/project/src/foo.cpp`	`tested file defined in bdf will be analyzed`
`/home/devel_1/project/includes/foo.h`	`#included by foo.cpp`
`/home/devel_1/project/includes/other.h`	`not #included by foo.cpp`
`/home/devel_1/common/common.h`	`#included by foo.cpp`

Assuming module *MyApp* is defined with `/home/devel_1/project root` location, the following files will be tested as part of the module:

`/home/devel_1/project/src/foo.cpp`	`belongs to MyApp as MyApp/src/foo.cpp; will be analyzed`
`/home/devel_1/project/includes/foo.h`	`belongs to MyApp as MyApp/includes/foo.h; will be analyzed`

/home/devel_1/project/includes/other.h	not #included; will not be analyzed
/home/devel_1/common/common.h	does not belong to MyApp; will not be analyzed

Defining a Basic Module Structure

Use the -[<MODULE_NAME>=]<MODULE_ROOT_LOCATION> switch to define a module. If the name is unspecified, the name of the root directory will be used:

```
-module MyApp=/home/devel_1/project
-module /home/devel_1/project
-module MyModule=../projects/module1
-module .
```

Alternatively, module structures can be defined in a custom configuration file using the `cpptest.scope.module.<MODULE_NAME>=<MODULE_ROOT_LOCATION>` property:

```
cpptest.scope.module.MyApp=/home/devel_1/project
cpptest.scope.module.MyModule=../projects/module1
```

Defining a Module with Multiple Root Locations

Add a logical path to the module name that points to the appropriate root location to define multiple, non-overlapping locations:

```
-module MyApp/module1=/home/devel_1/project -module MyApp/module2=/home/external/module2/src

cpptest.scope.module.MyApp/module1=/home/devel_1/project
cpptest.scope.module.MyApp/module2=/home/external/module2/src
```

Fine-tuning the Input Scope

Use the `-resource` switch to specify a file or set of files for testing.

```
-resource /home/cpptest/examples/ATM/ATM.cxx
-resource /home/cpptest/examples/ATM
-resource ATM.cxx
```

You can specify the following resources in the path:

- File path (only selected file will be tested)
- Directory path (only files from selected directory will be tested)
- File name (only files with selected name will be tested)

Use the -include and -exclude switches to apply additional filters to the scope.

- `-include` instructs C/C++test to test only the files that match the file system path; all other files are skipped.
- `-exclude` instructs C/C++test to test all files except for those that match the file system path.

If both switches are specified, then all files that match `-include`, but not those that match `-exclude` patterns are tested.

```
-include pattern
-exclude pattern
```

The `-include` and `-exclude` switches accept an absolute path to a file, with asterisk (`*`) as an accepted wildcard.

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

You can specify a file system path to a list file (`*.lst`) to include or exclude files in bulk. Each item in the `*.lst` file is treated as a separate entry.