

### 7.3 TESTING SELECTED FILES IN PROGRAM CONTEXT USING USESCAN

#### A. Introduction

Also by popular demand, Cleanscape has added the ability for users to test only a few files within the context of the entire program. This is useful for a large, stable sourcebase where perhaps 1-2 sourcefiles are being modified. Running an analysis over the entire sourcebase can be time- and resource-consuming, only to find (perhaps) a trivial error in the unit under test (UUT).

Users specify the UUT file(s), then use Component Test (GUI) or external program `usescan` (command line) to automatically determine the other source files necessary to resolve `USED` modules, `INCLUDE/#include` files, and preprocessor (`-D`) macros.

To accomplish this, there are two modes to `usescan`: a Definition Mode, where the project is described in an interactive command line session, and a Resolution Mode, where information is extracted from the project data obtained during Definition Mode.

#### B. Definition Mode

The program is invoked without any operands in a command line shell:

```
<install_dir>/bin/usescan (*nix)
"<install_dir>\bin\usescan" (Windows)
```

An interactive session begins, in which the user is asked to provide all the sourcefiles that comprise the project, the directories containing `INCLUDE` or `#include` files, and any preprocessor (`-D`) macros used in the program.

Upon conclusion, two files are created in directory `<install_dir>/projects`:

```
<projname>.cfp A project file listing all files, includes, and defines
<projname>.dep A recursive dependency list
```

A sample session follows. User input is in green. Note that the sample files are stored in the directory shown in the example: `<install_dir>/examples/heads`

```
READY (C:\cleanscape\flint\examples\heeds) c:\cleanscape\flint\bin\usescan
This interactive program creates a project description '.cfp' file for use by
CASAF or Flint. The goal is to have the test environment match the build en-
vironment as closely as possible, and provide contextual information when only
a portion of the entire sourcebase is being tested. There are 4 steps.

STEP 1: provide a single-word name for the project - no spaces, no extension..
heeds
Project file name is c:\cleanscape\flint\projects\heeds.cfp

STEP 2: enter ALL source filenames that comprise the project. Wildcards are
allowed. It may be useful to open a secondary command prompt or GUI-based file
manager to make this process easier. Gather info; press <ENTER> when ready...

Enter the first directory name containing project sourcefiles (or 'done'):
.

Enter the files in this directory that are part of the project, one per line.
Do NOT enter include files, #include files, or non-Fortran-source files.
Wildcards are allowed. When finished, enter 'done'.
*.f90
File(s) accepted. Enter next file, or 'done': done

Enter another directory name containing project sourcefiles (or 'done'):
done

STEP 3: enter include or #include paths. No need to relist source directories.
When done, enter 'done'.
Enter include or #include directory path #1:
done

STEP 4: enter any -D macros which are used in the build process, all on one
line, separated by spaces. Do not add the '-D', and do not use a space around
an '='. If there are no defines, enter 'none'.
Enter the list of -D macros:
none

Done! Project file is - c:\cleanscape\flint\projects\heeds.cfp
```

### C. Resolution Mode

#### GUI.

From the Flint GUI, select Component Test from the Misc Options tab in the lower left frame of the GUI (see Section 4.B.8). Necessary files are automatically added to the analysis, and any include directories or preprocessor macros are input to the analysis as well, if you specified them during project definition a la Part B above.

If you use preprocessor directives like #if or #include, be sure to

- Click the checkbox “Preprocessor” on the Source Config tab (and if necessary, specify the preprocessor binary by using the Locate... button at the bottom of that same tab).
- Specify any include directories for the Fortran INCLUDE directive or preprocessor #include statement. Skip this step if you have already provided this information during project definition Part B.

- Define or Undefine any preprocessor macros (symbols) necessary for proper branching on the Misc Options tab. Again, no need to perform this step if you already provided this information during project definition Part B.

When the analysis is run, there is no outward appearance that files have been added: the file list in the GUI remains the same, listing only the UUTs you originally specified. However, the top of the analysis report will show you which files have been analyzed.

*Example:* Run the usescan command line session as shown in Part B. Start the GUI, click the Add File button, select xmlio.f90 from the 'examples/heeds' subdirectory. Check the Component Test box on the Misc Options tab, then run the analysis. You will see that three files were analyzed, despite your having specified only one UUT:

```

1 Analysis Report | 2 Statistics | 3 Cross Reference | 4 Call Tree | 5 Include Tree
>>> Source analysis:
Directory c:\progra~2\cleanscape\flint\examples\heeds\
utilities.f90  university.f90  xmlio.f90
*****
Subroutine USAGE                               File utilities.f90      Line 326
      <Module subprog of UTILITIES>
*****
Subroutine CHECK_ARRAY_BOUND                   File utilities.f90      Line 357
      <Module subprog of UTILITIES>
*****
Subroutine LOG_COMMENT                         File utilities.f90      Line 389
      <Module subprog of UTILITIES>

```

## Command line.

The syntax of Resolution Mode is -

```
usescan -G -P <projfile> [-E <cmdfile>] <sourcefile1> [sourcefile2 ...]
```

where *profile* is the name you supplied during the interactive session (Definition Mode). No path or extension information is required.

Without **-E**, data is written as a space-separated string to stdout.

With **-E**, data is written one per line to the user-specified Flint command file (intended to be run with Flint's **-E** command line operand).

Errors are written to stderr. Return codes are set as follows:

**0** → success, **3-5** → warning, **8** → error, **10** → setup fail

**Example 1:** Having previously run `usescan` in Definition Mode as shown in Part B, run the command (from the 'examples/heeds' subdirectory)

```
usescan -G -P heeds xmlio.f90
```

to obtain the following, which has spaces between each datum (the output is one continuous line, but word wrapped in this document and in the shell):

```
-Ic:\cleanscape\flint\examples\heeds c:\cleanscape\flint\examples\
heeds\utilities.f90 c:\cleanscape\flint\examples\heeds\
university.f90 c:\cleanscape\flint\examples\heeds\xmlio.f90
```

**Example 2:** Same as above, but store the results in a Flint command file:

```
usescan -G -E \temp\heeds.cmd -P heeds xmlio.f90
```

The contents of `\temp\heeds.cmd` are as follows:

```
-Ic:\cleanscape\flint\examples\heeds
c:\cleanscape\flint\examples\heeds\utilities.f90
c:\cleanscape\flint\examples\heeds\university.f90
c:\cleanscape\flint\examples\heeds\xmlio.f90
```

**Example 3:** Specify a UUT that requires more modules:

```
usescan -G -P heeds server.f90

-Ic:\cleanscape\flint\examples\heeds c:\cleanscape\flint\main\
isobind.lsh c:\cleanscape\flint\examples\heeds\utilities.f90 c:\
cleanscape\flint\examples\heeds\university.f90 c:\cleanscape\flint\
examples\heeds\xmlio.f90 c:\cleanscape\flint\examples\heeds\html.f90
c:\cleanscape\flint\examples\heeds\editenlistment.f90 c:\cleanscape\
flint\examples\heeds\editstudent.f90 c:\cleanscape\flint\examples\
heeds\editchecklist.f90 c:\cleanscape\flint\examples\heeds\
editblocks.f90 c:\cleanscape\flint\examples\heeds\editsections.f90
c:\cleanscape\flint\examples\heeds\editteachers.f90 c:\cleanscape\
flint\examples\heeds\edituniversity.f90 c:\cleanscape\flint\examples\
heeds\initialize.f90 c:\cleanscape\flint\examples\heeds\server.f90
```

For Examples 1 and 3, copy/paste the output to a Flint analysis; for Example 2, run Flint with the `-E\temp\heeds.cmd` (you can add any other analysis controls you'd like to the `.cmd` file). Remember Flint option `-p` if your project requires running the code through a preprocessor before analysis!

To see a help listing from a shell prompt, run the following command:

```
<install_dir>/bin/usescan -?          (*nix)
"<install_dir>\bin\usescan /?"        (Windows)
```