

Cleanscape Fortran-lint User Interface Guide

Version 7.7x



Sales and Service Office

PO Box 616
Franklin Springs, GA 30639
Toll-free 800-944-LINT
Direct 706-245-1070
Fax 706-432-1720

www.cleanscape.net
sales@cleanscape.net
support@cleanscape.net

Note: Licensed users may photocopy for distribution.

**Direct comments concerning this manual to the address on the title page or
support@cleanscape.net**

Copyright © 1987-2024

**CLEANSCAPE
NOTICE OF COPYRIGHTS**

Copyrighted by Cleanscape as an unpublished work. All rights reserved. In claiming any copyright protection which may be applicable, Cleanscape reserves and does not waive any other rights that it may have (by agreement, statutory or common law, or otherwise) with respect to this material. See Notice of Proprietary Rights.

NOTICE OF PROPRIETARY RIGHTS

This manual and the material on which it is recorded are the property of Cleanscape. Its use, reproduction, transfer and/or disclosure to others, in this or any other form, is prohibited except as permitted by a written License Agreement with Cleanscape. Cleanscape reserves the right to update this document without prior notification.

FortranLint is a trademark of Cleanscape Software International.

All other trademarks and registered trademarks are the property of their respective owners.

NOTE: Some screenshots in this document may be of older versions. In such cases the salient feature(s) on that screen are unchanged in the newer version.

Table of Contents

PART I Introduction	5
1.1 WELCOME.....	5
1.2 DOCUMENTATION.....	5
1.3 PURPOSE.....	5
A. Function	5
B. Application.....	6
C. Advantages	6
D. Flow of Analysis	6
PART II Requirements, Installation, and Uninstallation	7
2.1 WINDOWS.....	7
A. System Requirements	7
B. Software Setup Procedure.....	7
C. Uninstallation	8
2.2 UNIX/LINUX.....	9
A. System Requirements	9
B. Software Setup Procedure.....	9
C. Uninstallation – manual process.....	9
PART III Activating Flint	11
A. Registration Process – Windows	11
B. Registration Process – Linux/Unix.....	11
PART IV Running the Flint GUI	13
A.0 Overview	13
A.1 Windows editors	13
A.2 Unix/Linux editors.....	13
A.3 Main GUI Window	13
A.4 Snapshot Window	16
B. Components.....	17
C. Creating a new project	21
D. Opening an existing project.....	21
E. Saving a project.....	22
F. Modifying a project	23
G. Execute test.....	25
H. Review reports	25
I. Online Help.....	30
J. Sub-Menu Functions.....	30
K. Operating the GUI using the Keyboard; Keyboard Shortcuts.....	32
L. Changing fonts / sizes.....	33
PART V Running Flint from the Command Line	35
A. Introduction	35
B. Operation	35
C. Return Codes.....	35
D. Command Line Example: Basic Analysis	36
E. Command Line Example: Call Tree.....	38
F. Command Line Example: Cross Reference.....	38
G. Command Line Example: Summary Report with Refactor Data	40
H. Other useful command line options and external utilities	41

PART VI Running Flint from IDEs	43
A. Overview	43
B. Installation	43
C. Operation	44
D. Uninstallation	46
PART VII Miscellaneous Information	47
7.1 ADDITIONAL STEPS FOR WINDOWS PERMISSIONS	47
A. Applicability	47
B. Details	47
7.2 ADDING AN EXTERNAL EDITOR TO THE GUI USING SETEDITOR	49
A. Introduction	49
B. Operation	49
7.3 FIND MISSING MODULES WITH USESCAN	51
Introduction	51
Generate USE dependencies	51
Obtain USE dependency list	53
usescan project creation alternatives	53
7.4 MINIMIZING FALSE POSITIVES FOR ENTIRE PROJECTS	55
Introduction	55
Step 0: gather the file list for the entire project	55
Step 1: check file list integrity	55
Step 2: check for missing procedures and unsupported Fortran	56
Step 3: run a full analysis – in stages	57
7.5 TESTING SELECTED FILES IN PROGRAM CONTEXT USING FILTFLINT	59
A. Introduction	59
B. Filter results from an existing Flint run	59
C. Run an analysis and immediately filter those results	60
D. Sample session using filtflint	60

PART I Introduction

1.1 WELCOME

Thank you for your product purchase! With Cleanscape Fortran-lint (Flint), you have the most powerful static source (lint) analysis available for Fortran 77→18 code. Flint in its command-line form has been assisting Fortran programmers for over a quarter century; the GUI is an ease-of-use enhancement to the venerable Flint product for a new generation of programmers – and anyone tired of command prompts or desiring the productivity gains available with a GUI.

As of version 7.7, there are 911 unique messages that can diagnose 1662 situations in Fortran code.

1.2 DOCUMENTATION

This is the “quick start” guide for the Flint static analyzer. There are three modes of Flint operation on Unix/Linux, and three on Windows:

A. *Cleanscape GUI*

B. *Command line*

C1. *Visual Studio integration using Cleanscape automation (Windows only)*

C2. *Xlint graphical browser (Unix/Linux only)*. This product remains under support, but the Flint GUI effectively supersedes its functionality. Detailed information is available starting in Section 12 of the [Flint Reference Manual](#) located in the ‘doc’ subdirectory, should you require this interface.

This document’s sole purpose is to describe these various interface modes. Flint is very rich in analysis controls and reporting; to gain maximum benefit from your product purchase, we urge you to read and keep handy the [Flint Reference Manual](#).

While on the topic of documentation: if you choose Cleanscape GUI, be sure to check out the Online Help facility! It’s concise yet useful information. The Table of Contents and many interrelated items in the help text are hyperlinked to make information access quick and easy.

New features in the latest release are marked in this document with **NEW v7.x**, with ‘x’ being a digit referring to a minor revision level.

1.3 PURPOSE

A. *Function*

1. Flint is a programming tool that simplifies the debugging and maintenance of both large and small Fortran programs. The Flint GUI provides ease-of-use enhancements to the venerable Flint command line product.
2. The Flint source code analyzer that can detect a wide range of problems, e.g.,
 - a. Inappropriate arguments passed to functions
 - b. Inappropriate library calls
 - c. Non-portable code
 - d. Type usage conflicts across different modules
 - e. Unused variables and dead code

B. Application

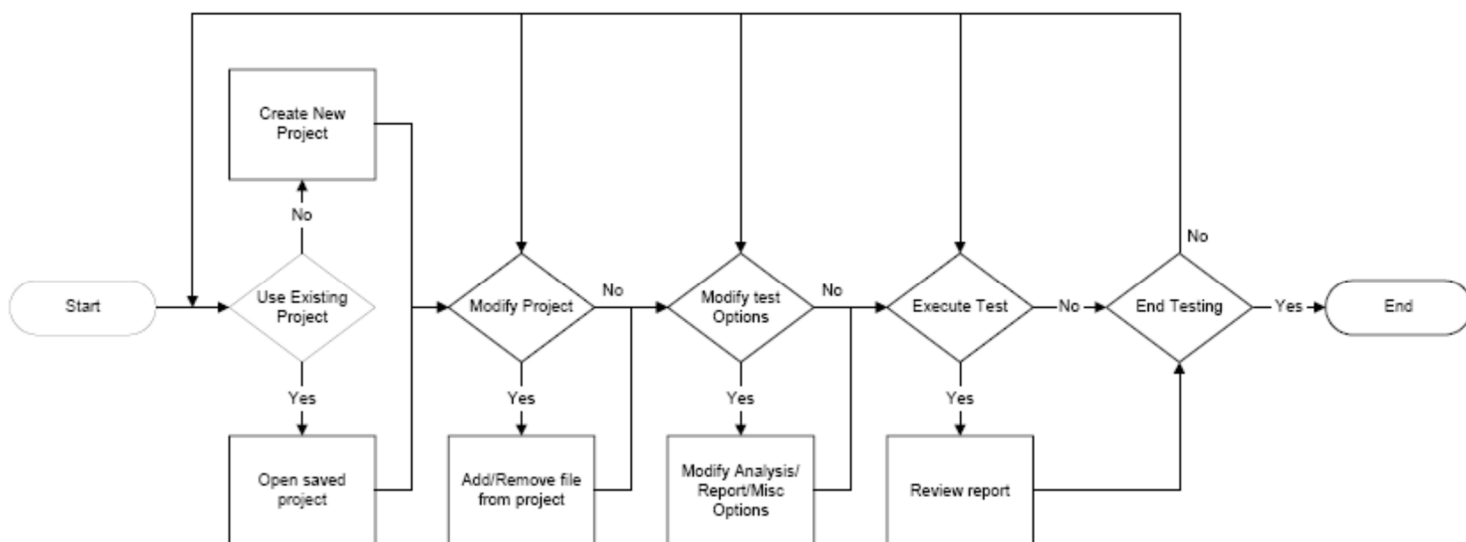
1. Flint can be used to:
 - a. Check source files before they are compiled
 - b. Isolate obscure problems
 - c. Identify problems before debugging is required

C. Advantages

1. The diagnostic messages produced by Flint are more detailed than those produced by standard compilers, and cover a much wider range of syntactic and semantic problems.
2. Flint analyzes source files both individually and as a group, and can therefore identify problems that are beyond the scope of a compiler – especially the global (program) scope.
3. Flint is effective in reducing development time and improves Fortran programming style.
4. Analysis and other report results are hyperlinked from the message to the related line of source using your favorite editor.
5. Cleanscape exclusive report content enhances your ability to comprehend and manage your program. These reports include call trees, cross reference for the entire program, USE trees, and include trees.

D. Flow of Analysis

1. The following flowchart illustrates the Flint test process:



PART II Requirements, Installation, and Uninstallation

Using FortranLint is subject to the terms and conditions contained in shrinkwrap_license.pdf, located in the 'doc' subdirectory. If you do not agree to the terms of that agreement, discontinue use of the associated software product immediately and contact sales@cleanscape.net within 15 days of purchase to arrange for return.

2.1 WINDOWS

A. System Requirements

1. Hardware

Any configuration sufficient to run Windows is sufficient for Flint.

2. Operating System

- a. Microsoft Windows 11
- b. Microsoft Windows 10
- c. Microsoft Windows 8
- d. Legacy support: Microsoft Windows 7, Microsoft Windows Vista®, Microsoft Windows XP® SP2, Microsoft Windows 2000® SP2, Microsoft Windows NT® 4.0 SP6a, Microsoft Windows 98® and 98® SE

3. Web Browsers

- a. Chrome® and Chromium-based browsers
- b. Firefox®
- c. Opera®
- d. Microsoft Edge

B. Software Setup Procedure

1. Installation

- a) Download `flintgui<ver>_win.exe` to a temporary directory, then run it.
- b) An installer window will appear and extract a number of files to the installation directory you specify (hereinafter referred to as `<install_dir>`; the default is `c:\cleanscape\flint`). The installer exits automatically, and no reboot is required, though you must close/reopen any command prompts.
The installer will upgrade you to x64 versions of key executables if applicable.
- c) If you are running the demo version of the product, no key is necessary. Otherwise, obtain and install a license key as described in Section 3.
- d) The installer searches for recent versions of Visual Studio on your machine and if found, automatically installs the Cleanscape tools. All installers are available in `<install_dir>\ideinstall`
Care has been taken to allow users who are not logged in as “owner” to successfully install the Visual Studio tools; if you encounter problems, reinstall as owner (or have your Administrator install) and notify us.

- e) Finally, the installer adds the `main` subdirectory to your system PATH – necessary for running Flint (or any of its associated support programs) from the command line. To do this manually, enter the following command:
- ```
set PATH=<install_dir>\main;%PATH%
```

## 2. Additional steps for user privileges / access control

If you're installing Flint as Administrator, and you want to make the program accessible to ordinary Users, some additional steps are required. For more information, see Section 7.1.

## C. Uninstallation

**NOTE:** You will need owner privileges if that is how the product was installed.

### 1. Manual uninstallation – *recommended*

- a) Delete the installation directory and its subdirectories.
- b) Delete the Flint GUI icon from the desktop
- c) Remove environment variable `FLINTHOME` and the Cleanscape directory from your PATH:
  - right click your “My Computer” icon on the desktop, then select “Properties”
  - click the “Advanced” tab or click “Advanced System Settings”
  - click the “Environment Variables” button and in the System Variables section:
    - delete the `FLINTHOME` line
    - double-click the text field “Path” in the System Variables area, and from that string, delete `<install_dir>\main`
- d) If Microsoft Visual Studio is installed on your machine, tools were automatically integrated upon Flint installation. Delete these tools using the following steps:
  - Open your Visual Studio IDE.
  - Select the Tools dropdown menu.
  - Select “External Tools...”
  - Click on each Cleanscape tool in turn, then click on the Delete button.

### 2. Using Windows System Restore

The installer created a system restore point just prior to installation. If you have not added new programs in the interim, you can safely roll your system back to this point.



## 2.2 UNIX/LINUX

### A. System Requirements

#### 1. Hardware

A minimum of 256 MB memory is required for Flint.

#### 2. Operating System. Note the GUI version may differ amongst the various hosts.

- a. Most GNU/Linux OSes, including RedHat®, SuSE®, Debian®, Ubuntu®
- b. Mac OS-X® Sonoma (Apple Silicon) and Mojave (x86)
- c. HP HP-UX®
- d. IBM AIX®
- e. SGI Irix®
- f. Sun Solaris®
- g. FreeBSD

#### 3. Web Browsers

- a. Firefox®
- b. Opera®
- c. Mozilla® or Netscape Navigator®

### B. Software Setup Procedure

**Installation – installation as root is easier and recommended. Refer to the installation notes for details. The ‘#’ below represents the root prompt.**

- a) **Download the latest version of flintgui<ver>\_<OS>.taz to a temporary directory, e.g., /tmp.**

**There is a 64-bit version of Flint for Linux and Mac.**

- b) **Create installation directory, e.g., /usr/local/cleanscape, and cd to it.**

- c) **Use the following command to extract the files.**

```
tar xzpvf /tmp/flintgui<ver>_<OS>.tar
```

**(For non-gnu tar, one would first run gunzip, then tar xpvf)**

- d) **If you are running the demo version of the product (Linux or Mac), no key is necessary and you can proceed to step f) below. Otherwise, obtain and install a license key as described in Section 3.**

- e) **If this is a server-based application, start the daemon on the server as root:**

```
startup
```

**NOTE:** The daemon must be running on the server before clients can access/ use the product. Rerun `startup` each time your server is rebooted.

- f) **If you intend to run Cleanscape Flint from the command line, create these environment variables (examples below are for sh/bash):**

```
export CSIAPPPBASE=<install_dir>
export FLINTHOME=$CSIAPPPBASE/flintgui.dir/main
export PATH=$CSIAPPPBASE:$FLINTHOME:$PATH
```

### C. Uninstallation – manual process

- a) **Delete the installation directory and its subdirectories.**
- b) **Delete .myeditor.lst, .ftemplate.csi, .flint.ini, and ./flint/\* from users' \$HOME directories.**



## PART III Activating Flint

**Note1:** If you are a demo user (Windows/Linux/Mac), *no key is necessary; proceed to the next page.*

**Note 2:** The license managers are different between Windows and Unix/Linux.

### A. Registration Process – Windows

You should have received a temporary key good for 30 days (the warranty period); if not, contact [sales@cleanscape.net](mailto:sales@cleanscape.net).

For most users, you simply copy the keyfile to %FLINTHOME%. Detailed instructions are provided in the email with the keyfile, as well as instructions for obtaining a permanent key.

There is practically no license management for single user licenses; floating license management is described in detail in doc\readme\_floating.txt.

### B. Registration Process – Linux/Unix

License registration is accomplished via the command line. You should have received a temporary key good for 30 days (the warranty period); if you didn't receive a tempkey, email [sales@cleanscape.net](mailto:sales@cleanscape.net).

0. Ensure you have set up the environment variables per the instructions in Section 2.2.B.f above.

1. Run the command, `flint activate`

Hit <Enter> to leave the number of license servers at its default of 1.

The next line from the activation program will contain your server code. Save this if you are requesting a permanent key (Step 3).

Enter your temporary key when prompted. Flint is registered and operational.

2. Run the command, `startup`

This starts the license manager daemon. Note: `startup` should be added to your server's initialization so it will restart with every server reboot; for assistance, contact [support@cleanscape.net](mailto:support@cleanscape.net).

3. To obtain your permanent key, email [sales@cleanscape.net](mailto:sales@cleanscape.net) and provide the server code displayed during step 1.

The next three sections describe in detail the operation of Flint

- from the GUI [Part IV](#)
- from the command line [Part V](#)
- integrated within IDEs [Part VI](#)

## PART IV Running the Flint GUI

### A.0 Overview

The Cleanscape GUI is a tried-and-true graphical interface used successfully for years. It is also the interface for our C/C++ offerings, and the planned interface for Java analyzers and test tools.

The Cleanscape GUI provides hyperlinking between the various reports (in the Reports frame) and the line of source in the source file that caused the message, using your favorite editor.

Advantages of the Cleanscape GUI include:

- Fast
- Easy to learn, navigate, and use
- Information readily at the programmer's fingertips
- Point-and-click control for options-laden Flint command-line product
- Access code at the relevant point using your favorite editor!

Supported code editors are listed below. It is also possible for users to integrate their own editor; see Section 7.2 for details on the `seteditor` program. For user-contributed editors, visit [http://www.cleanscape.net/products/contributed\\_editors.html](http://www.cleanscape.net/products/contributed_editors.html).

### A.1 Windows editors

- |                               |             |                      |
|-------------------------------|-------------|----------------------|
| • Atom                        | • GVim      | • UltraEdit          |
| • Crimson Editor              | • Notepad++ | • Visual Studio * #  |
| • Emacs                       | • Slickedit | • Visual Studio Code |
| • Epsilon Programmer's Editor | • Sublime   |                      |

### A.2 Unix/Linux editors

- |         |             |             |                      |
|---------|-------------|-------------|----------------------|
| • Atom  | • Nano *    | • Ultraedit | • Visual Studio Code |
| • Emacs | • Slickedit | • Vi *      |                      |
| • Joe * | • Sublime   | • Vim *     |                      |

\* Multiple instances of these editors will open with each link click.

# Version(s) of Visual Studio installed on the user's machine will be added automatically to the GUI dropdown during Flint installation.

### A.3 Main GUI Window

To start the GUI:

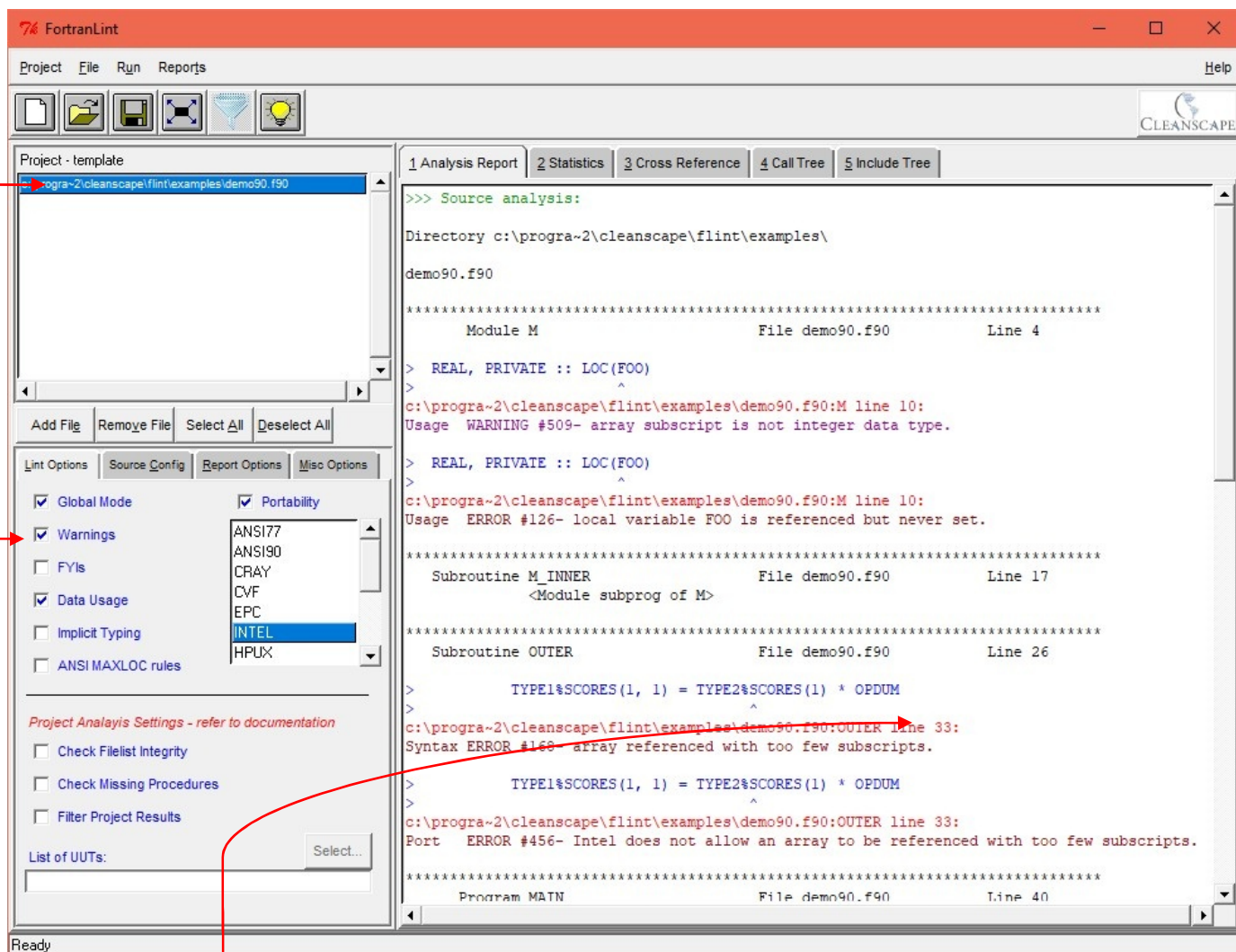
**Windows:** double-click the desktop shortcut created during installation.

**\*nix:** run the command, `$CSIAPPBASE/flintgui &`

All elements of the GUI are also controllable from the keyboard; this is discussed in [Section K](#) below.

The following screenshots depict a sample Flint session. The user has selected an example Fortran sourcefile and run the analysis. Also, they have chosen Visual Studio Code as their external code editor. How to take all these actions will be described later in this section.

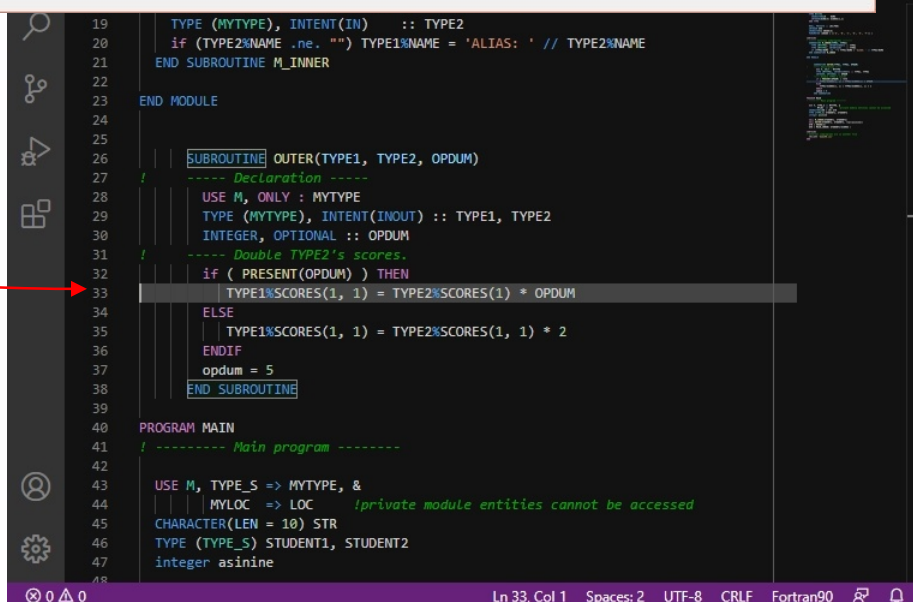
**NOTE:** All screen shots are Windows-based, but functionality is identical on Unix/Linux.



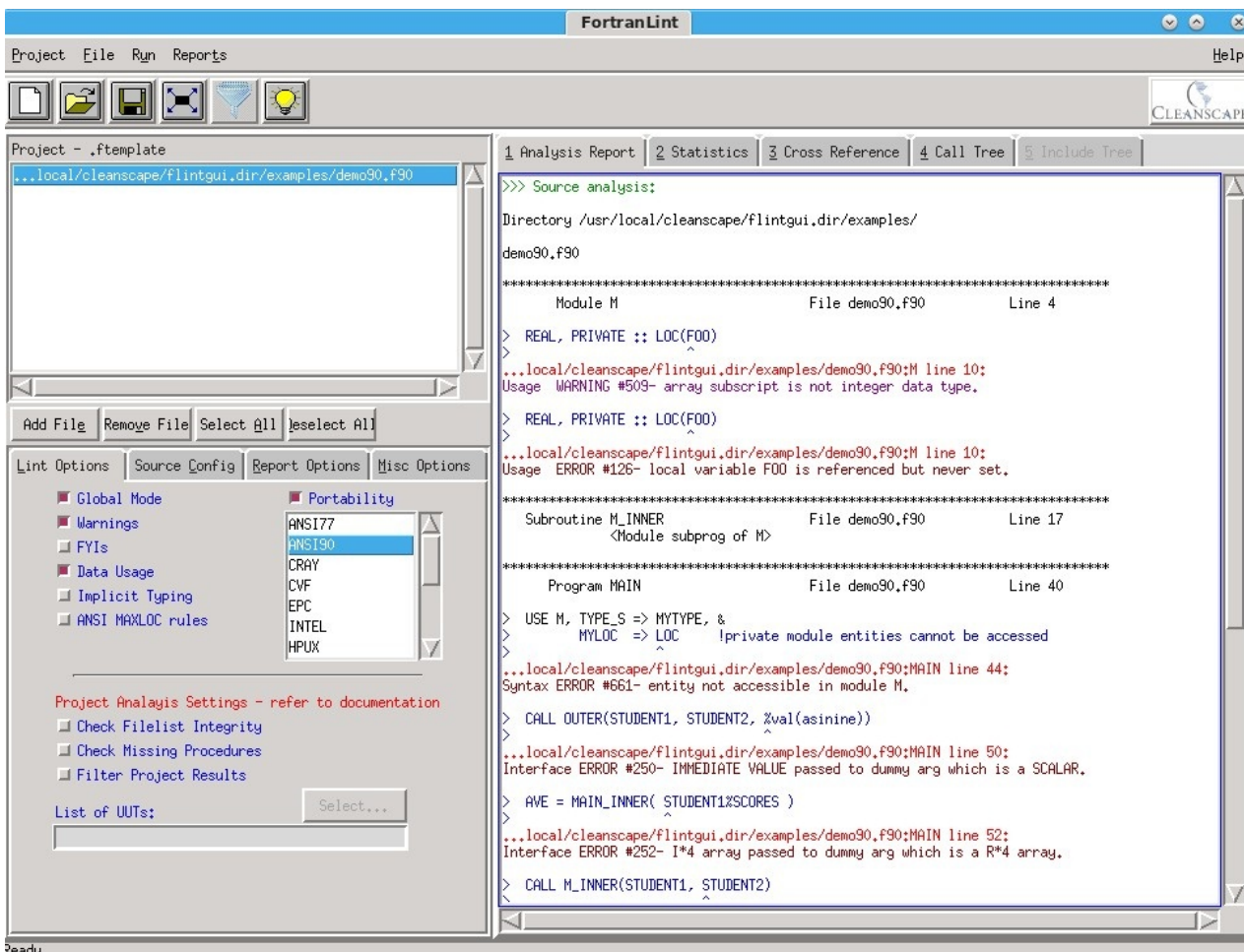
VS Code is activated when the red line containing "33" in the Analysis Report is left-mouse-clicked. Flint GUI positions VS Code to the line in the source file that caused the analysis result – line 33.

It is also possible to open any file listed in the Project window (upper left frame of the GUI) by right-mouse-clicking on the desired filename.

The Flint GUI remembers settings (e.g., checkboxes, include path, external editor – but not filenames) from the previous session by automatically creating a template file. Project files retaining filenames and settings can be saved and reopened in a later session.



The following screenshot is the Flint GUI running in Linux. Because the interface is identical – both in the GUI and the command line – there is no new learning curve for projects with a heterogenous workstation environment.





**NEW v7.6** A.4 Snapshot Window

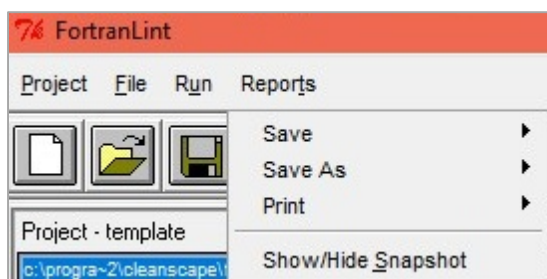
The Snapshot is a results summary window open to the right of the main GUI window. A new row is added every time an analysis is run in the current GUI session. Each row's background is color-coded, as is shown in this image:

- Red: errors were detected during the run
- Orange: warnings were detected during the run
- Yellow: FYIs were detected during the run
- Green: No messages produced during the run

A trend arrow for each level of severity indicates an increase, decrease, or no change in the quantity of errors, warnings, or FYIs between the current and last analysis run.

In this example, Flint has been run 6 times. The last (sixth) row has a green background, as the run was clear of any errors, warnings, or informational messages. Runs 1-5 contained errors (note the "Errs" count in column 1) and as a result, each such row in the Snapshot is red. (There are no arrows in row 1: being the first run, there are no trends yet.)

| Errs | Warn | FYIs |
|------|------|------|
| 4    | 0    | 1    |
| 5    | 0    | 0    |
| 4    | 0    | 1    |
| 5    | 0    | 0    |
| 4    | 0    | 1    |
| 0    | 0    | 0    |



Snapshot can be hidden or shown by clicking "Show/Hide Snapshot" under the Reports tab in the main GUI window. This functions as a toggle, hiding or showing Snapshot depending on its current status. Also, clicking this will open a new Snapshot if it had been closed previously.

Hiding Snapshot here is temporary; to close it for this GUI session, use *Close* – see next.

Similarly, right-clicking in the Snapshot window will open a popup menu to Hide or Close the window:

- *Hide* closes the window temporarily; it will reappear on the next run, and re-display prior runs and trends.
- *Close* will do just that: close the window and it does not reappear for the rest of the GUI session. However, it can reopened by clicking "Show/Hide Snapshot" under the Reports tab in the main GUI window, as described above.



Snapshot will reappear the next time the GUI is started and an analysis is run; to prevent its use entirely, edit file `%FLINTHOME%\flint.ini` or `$FLINTHOME/.flint.ini` to change the '1' to a '0' next to "Show results window", or '0' to '1' to allow its use once again.



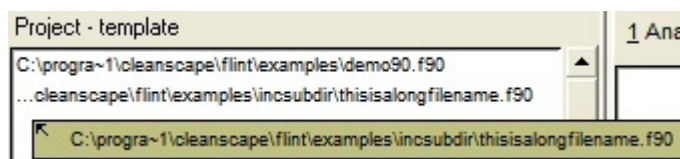
## B. Components

Where possible, each component features “balloon” help which will appear if you hover the mouse over an item or control description. Additional help for each item may be found in the Online Help (see [Section 4.I](#)).

1. Program menu: 

2. Shortcut bar: 

3. Project window:



Any file listed in the Project window can be opened in the selected editor by right-mouse-clicking the filename. Any filenames too long to fit the window are shortened to ~60 characters and an ellipsis is prepended. The full filename appears in a balloon tip if hovering the mouse over the name, as shown above.

4. Project shortcut buttons: 

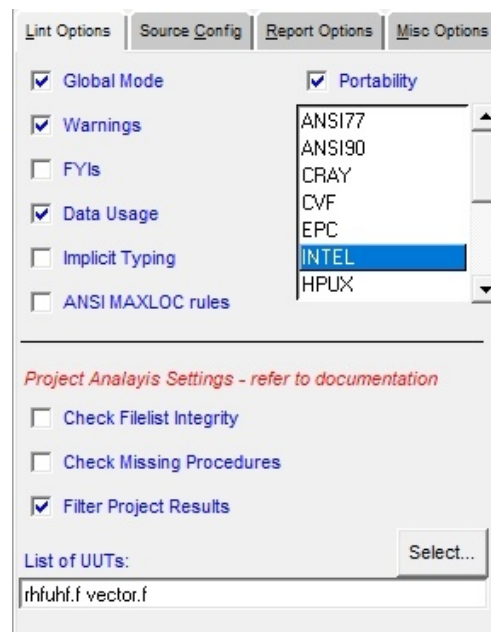
5. Lint Options tab (with “Portability” listbox activated). The GUI’s point-and-click control makes using the options-laden Flint command-line product easy!

Flint provides 12 portability options to help determine issues porting your code to different hosts. The ANSI77 and ANSI90 options are the most commonly used.

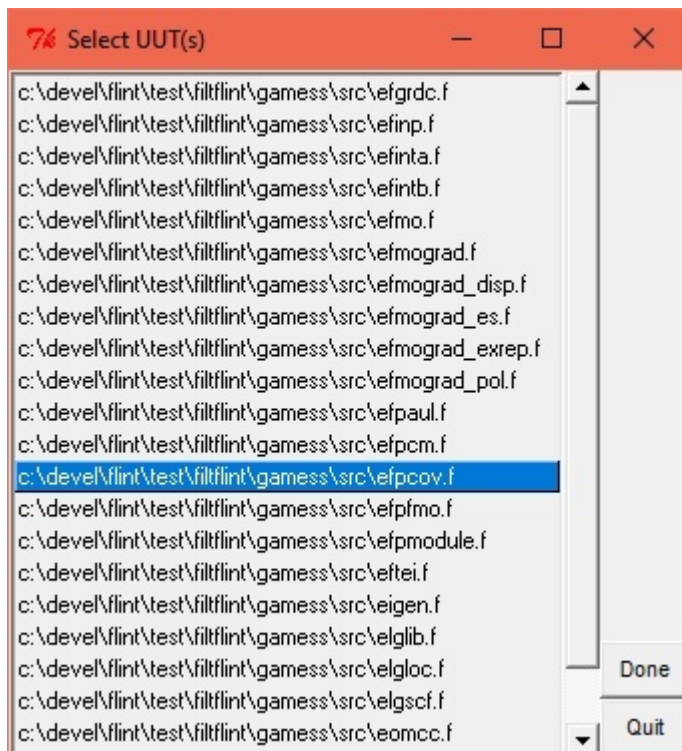
### NEW v7.6

Three project analysis settings are available:

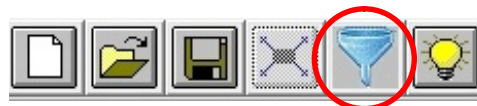
- Check Filelist Integrity* can eliminate scores of false positives due to missing modules or include files. It also checks for Fortran statements out of order (e.g., declarations in the wrong place) and will report if 132-column source is detected.
- Check Missing Procedures* reports missing functions or subroutines. This can mean additional files need to be added to the project filelist, or that it would be beneficial to create “stubs” for procedures for which there is no source code (e.g., library routines). To build stubs, see Section 9 of the [Flint Reference Manual](#).



- NEW v7.6** c) *Filter Project Results* can dramatically reduce the amount of output on large projects, allowing the user to focus on just one or a few files of interest, rather than the results for the entire program. In very large test cases, the total number of report lines is reduced by >99%! To use filtering,
- Select all the files comprising the program.
  - Run the two pre-analysis tests (Check Filelist Integrity and Check Missing Procedures). Utilizing these tests will decrease false-positive count.
  - Enable the “Filter Project Results” checkbox.
  - Press the Select... button. A dialog box will appear like the one below.



- Click on up to 5 files of interest (Units Under Test, or UUTs), then press Done
- Press the funnel icon on the Shortcut Bar, or select Run – Analyze+Filter from the Program Menu. A full analysis will be run, followed by the execution of external program `filtflint`.
- Upon completion, a set of filtered reports will be presented in the right frame of the GUI!



**NOTE:** Filtering runs a Flint analysis on the entire project, then invokes external program `filtflint` to filter the full set of results. This process can take several seconds or perhaps a couple minutes.

For managing large projects in Flint GUI, see Section 7.3.  
For more on `filtflint`, see Section 7.4.

## 6. Source Config tab.

“Dialect” is analogous to “Portability”; an example in English is, “Tell Flint that the incoming source was written for a Solaris compiler (dialect) and I want to know issues porting to a Lahey compiler (portability)”.

Also note the preprocessor option: if checked, Flint will search for `cpp` in your PATH; you can define a new path and/or preprocessor name (e.g., the `fpp` that came with your Fortran compiler) in the textbox at bottom.

The Source Config tab contains the following options:

- ☐ Debug Lines
- ☐ 132 Columns
- ☐ HPF directives
- ☒ Preprocessor
- ☐ Two-byte INTEGERS
- ☐ Ignore INCLUDE paths
- ☐ Ignore VMS logicals
- ☐ OpenMP directives

Language: Automatic (dropdown)

Dialect: Automatic (dropdown)

Source Format: Automatic (dropdown)

"Include" directories: [Browse...]

Preprocessor location, if not in PATH: [Locate...]

## 7. Report Options tab

Examples for all reports in Section 4.H below.

**NEW v7.0** Include Tree: display the inclusion of header files from Fortran `INCLUDE` lines and preprocessor `#include` directives.

**NEW v7.6** Assessment: in the Statistics report, list each procedure name, line count, parameter count, and cyclomatic complexity `v(G)`. These results may be sorted and are hyperlinked in the Statistics report; see example in Section 4.H.

In addition, informational message 909 will be emitted if `v(G)`, line count, or parameter count exceed the maximum specified on this tab. An example of this message is shown in the Analysis report, Section 4.H.

The Report Options tab contains the following options:

- ☒ Cross-Reference
- ☒ Call Tree
- ☒ Free Form
- ☒ Condense Tree
- ☐ Tabular
- ☐ Squash Tree
- ☒ INCLUDE Tree
- ☒ Trim Tree
- ☒ Statistics
- ☒ Assess, sort by: Parameter Count (dropdown)

Generate message if assessment exceeds (0=no msg):

`v(G)`: 4

Line Count: 0

Parm Count: 3

External Editor: Crimson Editor (dropdown)

Editor location: [Locate...]

c:\progra~2\Crimso~1

**Release Note:** For version 7.4 and later, the USE Report has been temporarily removed while under rewrite. If you need this report, contact [sales@cleanscape.net](mailto:sales@cleanscape.net).

## 8. Miscellaneous Options tab:

**NEW v7.6** Magic Comments: Processes magic comments inside source code. Presently supports only Omit (-O) to suppress a Flint message on the following sourceline. See Section 3.1.3 (under -M) of the Flint Reference Manual for details, including format of the magic comment inside the source file. Notification of magic comment use is provided in both the Analysis and Statistics reports as shown in Section 4.H.

Preprocessor defines/undefines can be specified in the first two textboxes on this tab, separated by spaces.

Call tree root is an alternative starting point rather than the main program; see Section 7.4.1 of `flintman.pdf` (located in the 'doc' subdirectory) for details.

Because Flint tracks all variables (even loop counters), cross references can rapidly become a very large. To offset this, Flint provides a filtering capability; see Section 8.3 of `flintman.pdf` for details.

An example filter shown here suppresses loop counters of a single letter.

Individual analyses can be enabled/ disabled by number in the appropriate text boxes on this tab.

Advanced Feature: Dataflow analysis can help identify problems with initialization, improper sequencing of set/reference instances, and identifying dead or wasteful code.

For example, with this option enabled, Fortran-lint will inspect both branches of an IF-THEN-ELSE conditional to determine if a variable has been initialized (set) in both branches, and whether a variable has been referenced before set in either branch.

Using this option will add processing time to the analysis. The amount of extra time is determined by the complexity of the source, i.e., nested loops, IF blocks or excessive use of GOTOs. Indeed, some runs could take years! *So dataflow analysis should be used sparingly and then only by senior programming staff.*

Release Note: For version 7.4 and later, Component Test has been temporarily removed while under rewrite. Its capability is available from the command line; if you need this report, contact [sales@cleanscape.net](mailto:sales@cleanscape.net).

The screenshot shows the 'Misc Options' tab with the following settings:

- ☒ Process Magic Comments
- Define symbols: [empty text box]
- Undefine symbols: [empty text box]
- Call Tree Roots: [empty text box]
- Cross Reference Filters: no.loop.named.? no.unreferenced.parameters no.unused.
- Disable these messages: 76 207 261 176
- Enable these messages: [empty text box]
- ADVANCED option - refer to documentation!
- ☐ Data-flow Analysis

## 9. Report windows:

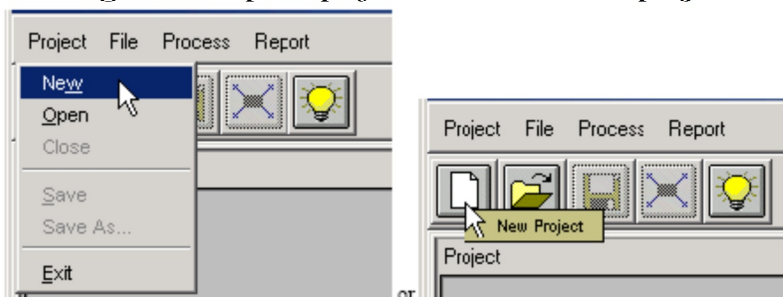
The screenshot shows five buttons for report windows:

- 1 Analysis Report
- 2 Statistics
- 3 Cross Reference
- 4 Call Tree
- 5 Include Tree

Example reports appear in [Section H](#) below.

### C. Creating a new project

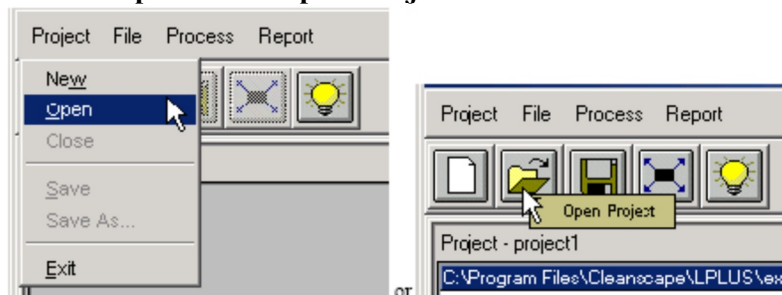
1. To create a new project, select Project/New from the menu or press the New Project button on the shortcut bar. Note: If a project is already open, a dialog box will prompt you to save the old project first.



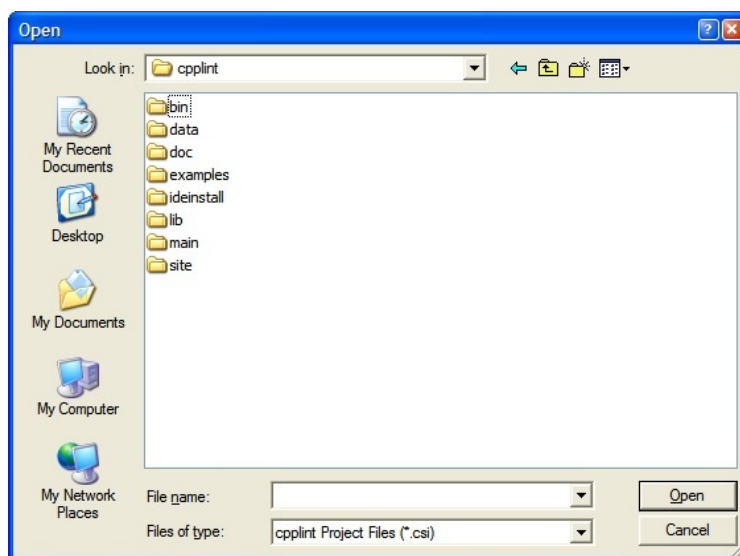
2. A new project name appears in the title, which can be saved to any desired name later.

### D. Opening an existing project

1. To open an existing Cleanscape GUI project, select Project/Open from the menu or press the Open Project button on the shortcut bar:

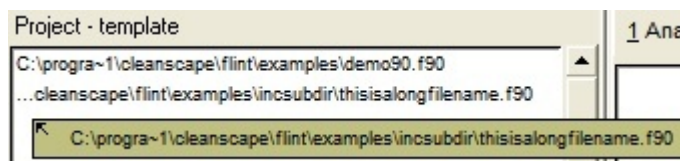


2. A standard Open dialog box will appear:
  - a. Browse to find/select a project file (with extension .csi).
  - b. When ready, press the Open button in the lower right corner.

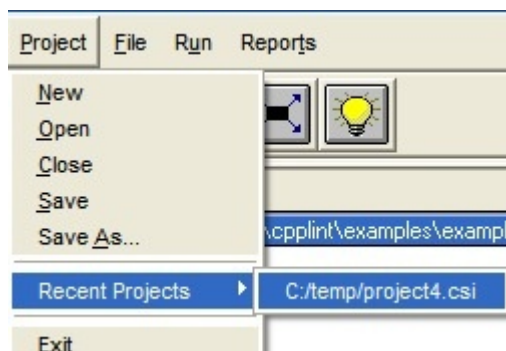




- Files associated with the project are displayed in the Project window:

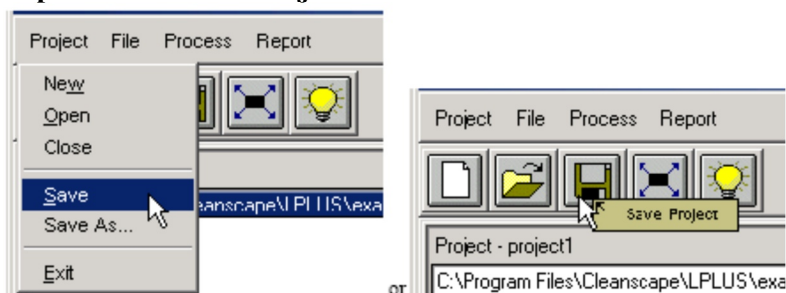


- It is also possible to open recent projects using the Recent Projects menu:



### E. Saving a project

- To save the current state of a project, select Project/Save from the menu or press the Save Project button on the shortcut bar:

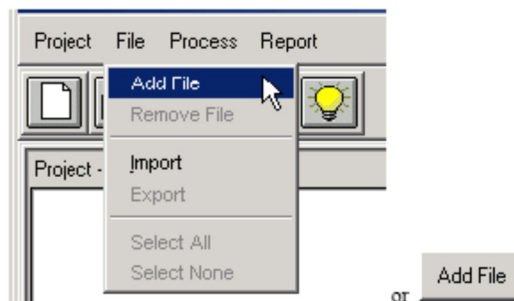


- If this is a new project, the Save As window will appear.
  - Enter a name for the project.
  - When done, press the Save button.
  - You can also use the "Save As..." feature in the Project dropdown to save an existing project under a new name.

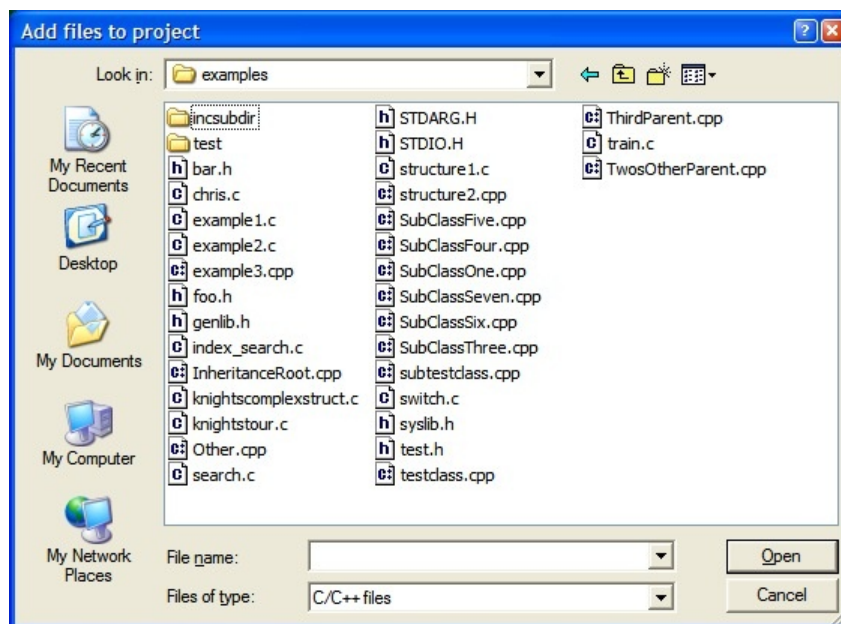
## F. Modifying a project

### 1. Add files to a project

- a. To add one or more files to a project, select **File/Add File** from the menu to add files into the project or press the **Add File** button on the project shortcut bar:



- b. The Add file window will appear:

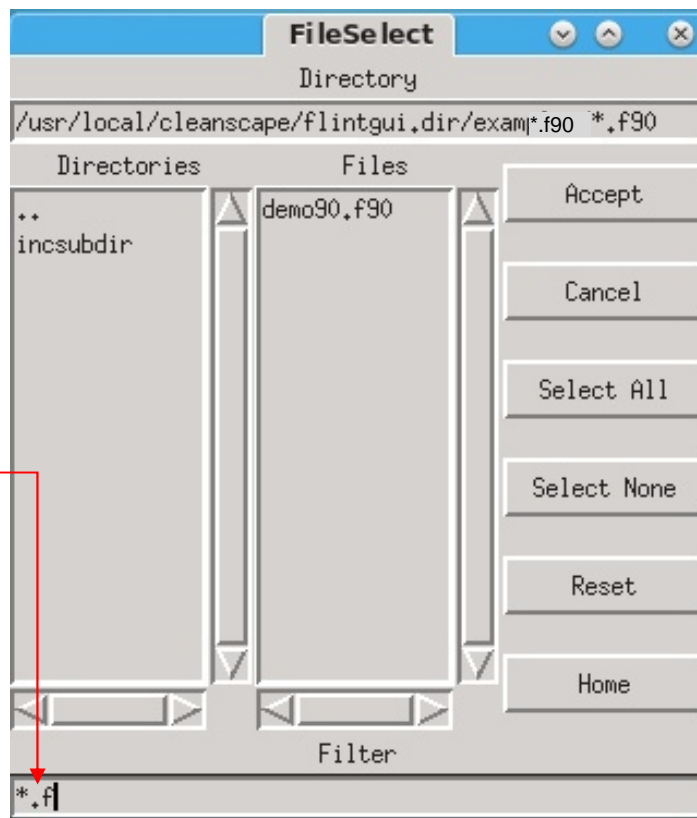


- c. For the Flint GUI, Fortran source files will be the default file type (.f, .f90).

- e. **UNIX NOTE:** The default file type is .f90, which can be modified by entering the appropriate type (e.g., \*.F) in the Filter textbox at the bottom of the dialog.

It is also possible to permanently modify the filter type by editing the "Default Add File filter" line in text file

\$HOME/.flint.ini.

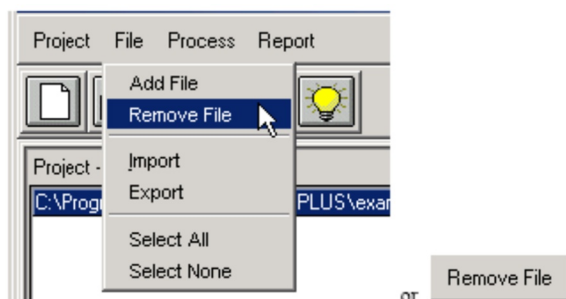


- e. **Multi-file selection:**

- 1) The file-selection dialog supports multiple-file selection under both MS-Windows and \*nix.
- 2) To add multiple files individually, use <Control> + Left Mouse Button. Each selected file will be highlighted.
- 3) To add a group of files:
  - (i) Left-click on the first file.
  - (ii) Hold down the <Shift> key.
  - (iii) Click the last file. The first, last, and all in-between will be highlighted.
  - (iv) When done, press the Open (Windows) or Accept (\*nix) button.

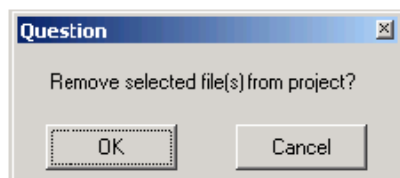
## 2. Removing files from a project

- a. To remove individual source files from a project, select the files to be removed, and then press the Remove File button. To remove all files from a project (i.e., to clear the file list), first press Select All, and then press the Remove File button.





- b. Press the OK button to confirm the removal operation:



- c. The updated file list is displayed in the project window.
- d. Note that this operation has no effect on the actual file on-disk.

#### G. Execute test

1. Create a new project or open an existing project for testing.

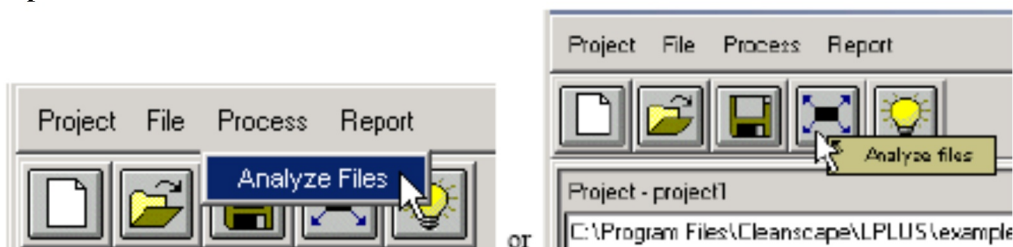
To create a new project, see [Section 4.C](#).

To open an existing project, [Section 4.D](#).

2. Select the files to be analyzed as explained in [Section 4.F.1](#).

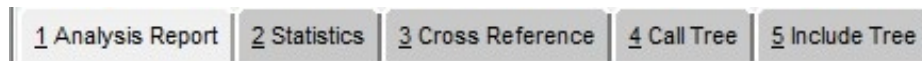
3. Modify options as necessary, using the tabs in the lower left frame of the GUI, as displayed in [Sections 4.B.5-8](#). See balloon help, Online Help, and the [Flint Reference Manual](#).

4. To analyze the selected files, use Process/Analyze Files from the menu or press the Execute test button on the shortcut bar:



#### H. Review reports

1. To view the generated reports, click on the appropriate report tab:



2. To print reports, or to save them to disk, use the Report menu dropdown at the top of the screen. Reports may be printed or saved collectively or individually.
3. Samples of each of the reports are depicted below. Remember that clicking any entry in **red** will open the source file at the appropriate source line in the specified External Editor.

Release Note: For version 7.4 and later, the USE Report has been temporarily removed while under rewrite. If you need this report, contact [sales@cleanscape.net](mailto:sales@cleanscape.net).

|                   |              |                   |             |                |
|-------------------|--------------|-------------------|-------------|----------------|
| 1 Analysis Report | 2 Statistics | 3 Cross Reference | 4 Call Tree | 5 Include Tree |
|-------------------|--------------|-------------------|-------------|----------------|

```

> AVE = MAIN_INNER(STUDENT1%SCORES)
> ^
c:\progra~2\cleanscape\flint\examples\demo90.f90:MAIN line 52:
Interface ERROR #252- I*4 array passed to dummy arg which is a R*4 array.

> CALL M_INNER(STUDENT1, STUDENT2)
> ^
c:\progra~2\cleanscape\flint\examples\demo90.f90:MAIN line 49:
Usage ERROR #126- local variable STUDENT2 is referenced but never set.

> CALL OUTER(STUDENT1, STUDENT2, %val(asinine))
> ^
c:\progra~2\cleanscape\flint\examples\demo90.f90:MAIN line 50:
Usage ERROR #126- local variable ASININE is referenced but never set.

> STR = GRADE(3)
> ^
c:\progra~2\cleanscape\flint\examples\demo90.f90:MAIN line 51:
Usage WARNING #127- local variable STR is set but never referenced.

Function MAIN_INNER File demo90.f90 Line 56
 <Internal subprog of MAIN>

=== Skipping msgs 28,3 due to magic comment for line 13 demo90.inc

c:\progra~2\cleanscape\flint\examples\demo90.f90:MAIN_INNER line 56.56 (demo90.inc):
Syntax FYI #909- Assessment v(G) exceeds user-specified limit (5/4).

Global checking:

Usage ERROR #742- module entity referenced but not set: M:FOO
Usage WARNING #743- module entity set but not referenced: M:AVE

```

**NEW v7.6** Note the Assessment section with the hyperlinked procedure name. Parameter Count is in green, indicating that it is the sort key.

| 1 Analysis Report | 2 Statistics | 3 Cross Reference | 4 Call Tree | 5 Include Tree |
|-------------------|--------------|-------------------|-------------|----------------|
|-------------------|--------------|-------------------|-------------|----------------|

```

>>> Statistics:

Number of source files: 1

Source files: 57 lines, 1350 bytes (18% comments, 82% code)
Include files: 19 lines, 396 bytes (5% comments, 95% code)
Total parsed: 76 lines, 1746 bytes (15% comments, 85% code)

Total subprograms: 5
 Subroutines: 2
 Functions: 1
 Program: 1
 Block Data: 0
 Modules: 1

Program Unit Assessments

PROCEDURE NAME TYPE LSTART LINECT PARMCT v(G)
c:\progra~2\cleanscape\flint\examples\demo90.f90::OUTER (SUB) 26 13 3 2
c:\progra~2\cleanscape\flint\examples\demo90.f90::M_INNER (SUB) 17 5 2 2
c:\progra~2\cleanscape\flint\examples\demo90.f90::MAIN_INNER (FCN) 56 1 1 5
c:\progra~2\cleanscape\flint\examples\demo90.f90::MAIN (PRG) 40 15 0 1

Individual message summary

Usage ERR #126- 3x: local * * is referenced but never set.
Usage WARN #127- 1x: local variable * is set but never referenced.
Syntax ERR #168- 1x: array referenced with too few subscripts.
Port ERR #206- 1x: %VAL is not supported by *.
Intrfc ERR #250- 1x: * passed to dummy arg which is * *.
Intrfc ERR #252- 1x: * array passed to dummy arg which is a * array.
Port ERR #456- 1x: * does not allow an array to be referenced with too
 few subscripts.
Usage WARN #509- 1x: array subscript is not integer data type.
Syntax ERR #661- 1x: entity not accessible in module *.
Usage ERR #742- 1x: module entity referenced but not set: *, *
Usage WARN #743- 1x: module entity set but not referenced: *, *
Syntax FYI #909- 1x: Assessment * exceeds user-specified limit (*/*).

Total messages: 13
INFO: 1 message has been suppressed through use of magic comments.

 Errors Warnings FYIs

Syntax: 2 0 0
Global Interface: 2 0 <supp>
Data usage: 4 3 <supp>
Intel port: 2 0 0

Implicit typing: <supp>

=====
FORTTRAN-lint Rev 7.58 18-Jun-2023 17:58:49
Default options: -O276 -Xno.named.IEEE_*,no.named.C_*,no.named.ISO_*
 -Xno.named.COMPILER_*
Expanded options: --2 --p -Ic:\progra~2\cleanscape\flint\examples
 -Ic:\temp\temp --e --d --a -PINTEL --m -w --f -g -u -O76
 -O207,261,176 -s -Asp,v4,p3 -Ttrim,nosquish,condensed -t
 -Xno.loop.named.?,no.unreferenced.parameters
 -Xno.unused.common.variables,linenumbers -x -W140 -Y250
 -Sc:\PROGRA~2\cleanscape\flint\main\reports\flint
 -Mmagicshow,depend

```

1 Analysis Report 2 Statistics 3 Cross Reference 5 Include Tree

```

*** Records:

STUDENT1 : type TYPE_S : local
 in (demo90.f90:MAIN) is 44-D 46-SA 47-SA 49-RA
STUDENT2 : type TYPE_S : local
 in (demo90.f90:MAIN) is 44-D 46-RA 47-RA
TYPE1 : type MYTYPE : local
 in (demo90.f90:M::M_INNER) is 16-P 17-D 19-S
 in (demo90.f90:OUTER) is 25-P 28-D 32-S 34-S
TYPE2 : type MYTYPE : local
 in (demo90.f90:M::M_INNER) is 16-P 18-D 19-R
 in (demo90.f90:OUTER) is 25-P 28-D 32-R 34-R

*** Vars/Arrays:

AVE : I*4 : public entity of module M
 in (demo90.f90:M) is 10-D
 in (demo90.f90:MAIN) is 49-S
DUM (:,:) : R*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)3-P
 (demo90.inc)4-D
 (demo90.inc)6-RA
 (demo90.inc)7-RA
 (demo90.inc)9-R
FOO : R*4 : public entity of module M
 in (demo90.f90:M) is 9-RB
I : I*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)6-RS
 (demo90.inc)9-R
J : I*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)7-RS
 (demo90.inc)9-R
LOC (adj) : R*4 : private entity of module M
 in (demo90.f90:M) is 9-D
OPDUM : I*4 : local
 in (demo90.f90:OUTER) is 25-P 29-D 31-RA 32-R
STR : CHAR*10 : local

```

1 Analysis Report 2 Statistics 3 Cross Reference 4 Call Tree 5 Include Tree

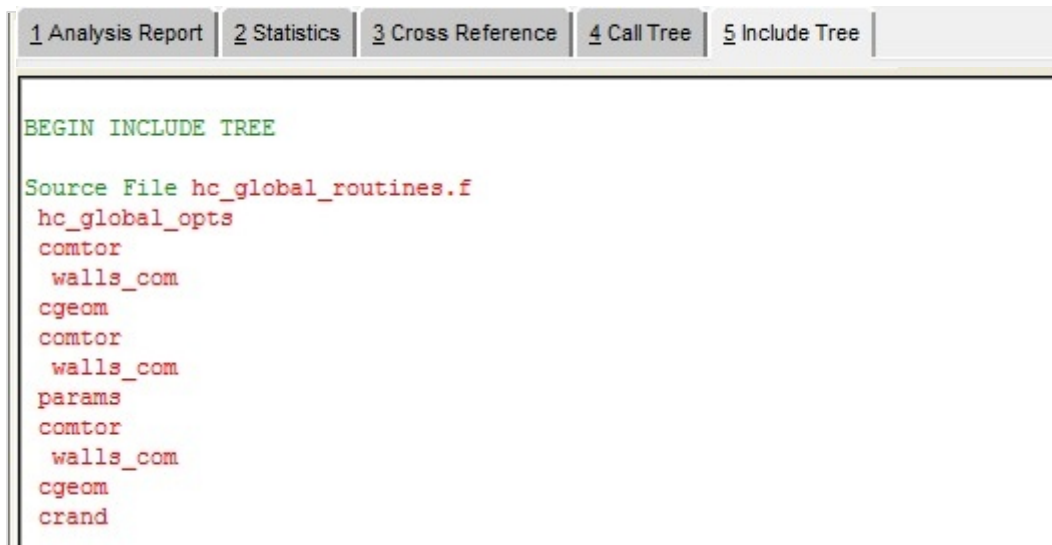
This is a primary tree starting at the program 'PROC DAT'

```

PROC DAT--+-GETUNIT
 |
 +-READNAME
 |
 +-SETTYPE--PRINT (1)--PRINTIT--+-DIPSTAT---*PRINT*
 | |
 | +-GETUNIT
 |
 +-PRINT see 1

```

**NEW v7.0** Include Tree. Flint scans both INCLUDE lines and #include directives.



```
BEGIN INCLUDE TREE

Source File hc_global_routines.f
 hc_global_opts
 comtor
 walls_com
 cgeom
 comtor
 walls_com
 params
 comtor
 walls_com
 cgeom
 crand
```

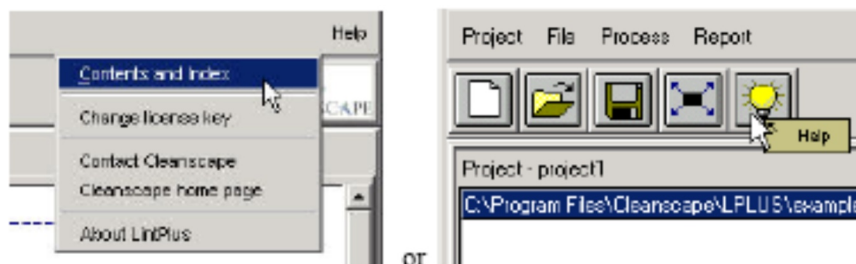


## I. Online Help

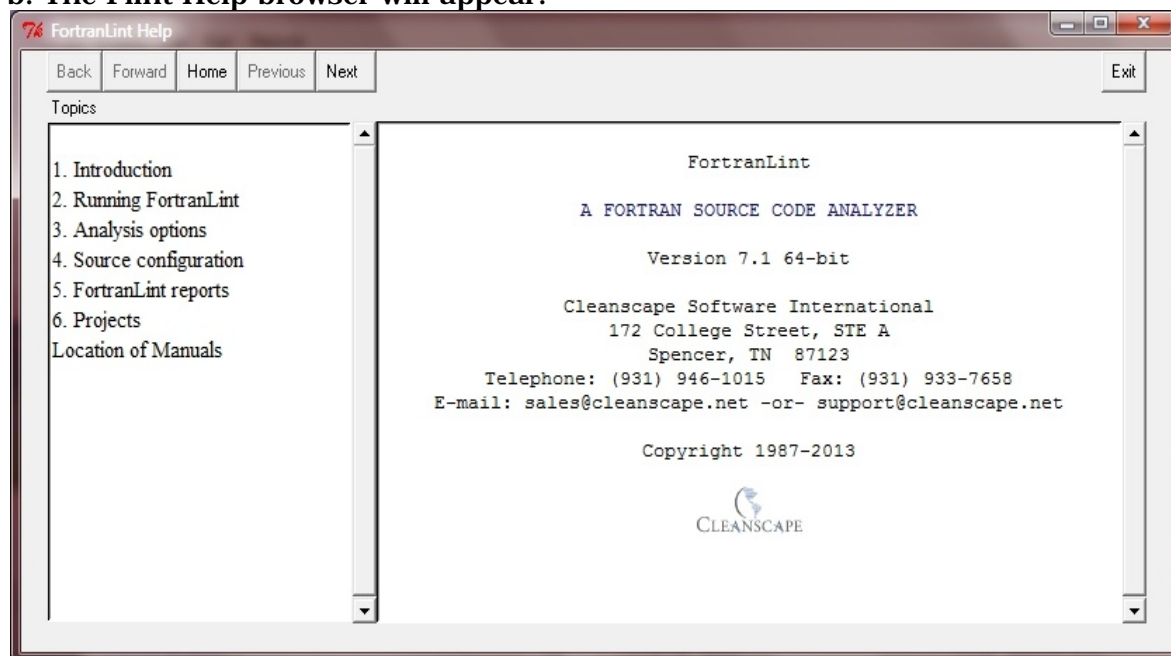
The Online Help System contains concise yet useful information for running the Cleanscape GUI. The Table of Contents and many interrelated items in the help text are hyperlinked to make information access quick and easy.

### 1. Accessing the Help System

- a. To access the online help system, select Help/Contents and Index from the menu or press the Help button:



- b. The Flint Help browser will appear:

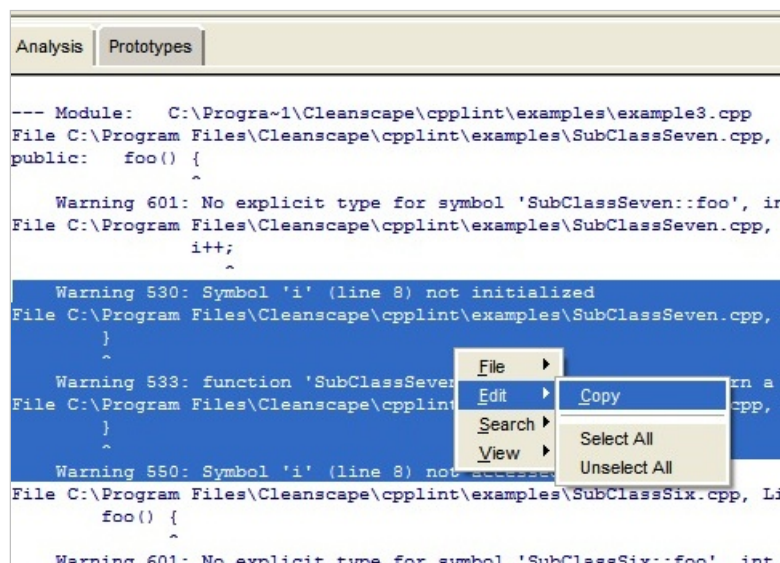


## J. Sub-Menu Functions

There are several “right-mouse-click” options available while in the Reports frame on the right hand side of the GUI. These features should be self-explanatory for those familiar with graphical environments. The more commonly used features are shown in detail below.

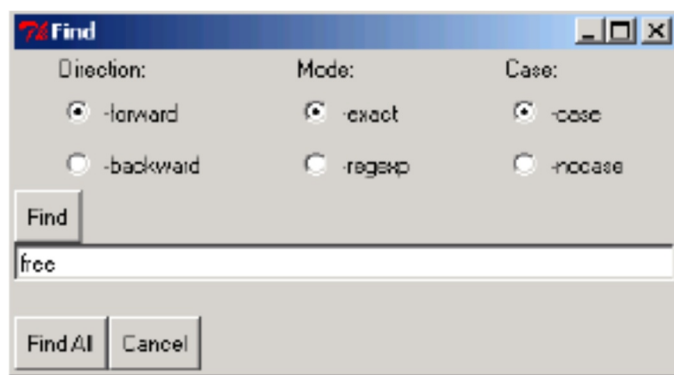
### 1. Copy

- a. Click and hold the left mouse button while dragging to select text.
- b. Press the right mouse button inside reports frame
- c. Select Edit -> Copy
- d. The text can now be pasted into other applications (e.g., Microsoft Word).



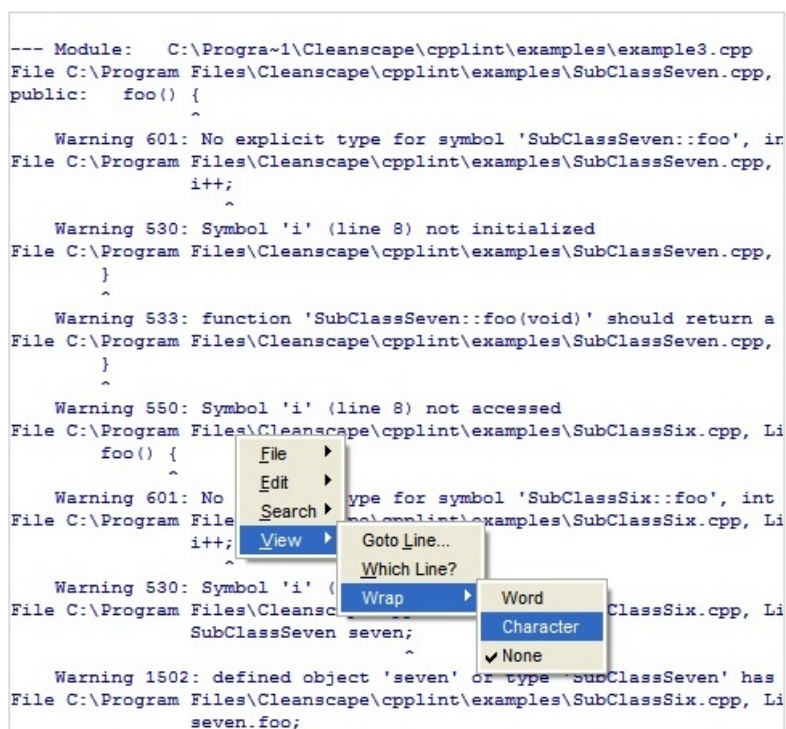
## 2. Search

- Press the right mouse button inside a report frame.
- Select Search -> Find.
- Enter string to search and select the desired options:
- The search result(s) will be highlighted.



## 3. Line Wrap

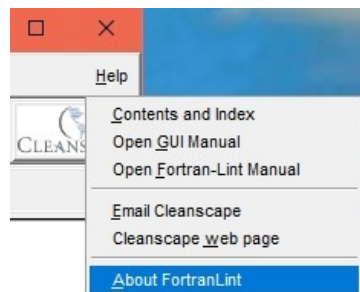
- Press the right mouse button inside a report frame
- Select View -> Wrap. The default is None.



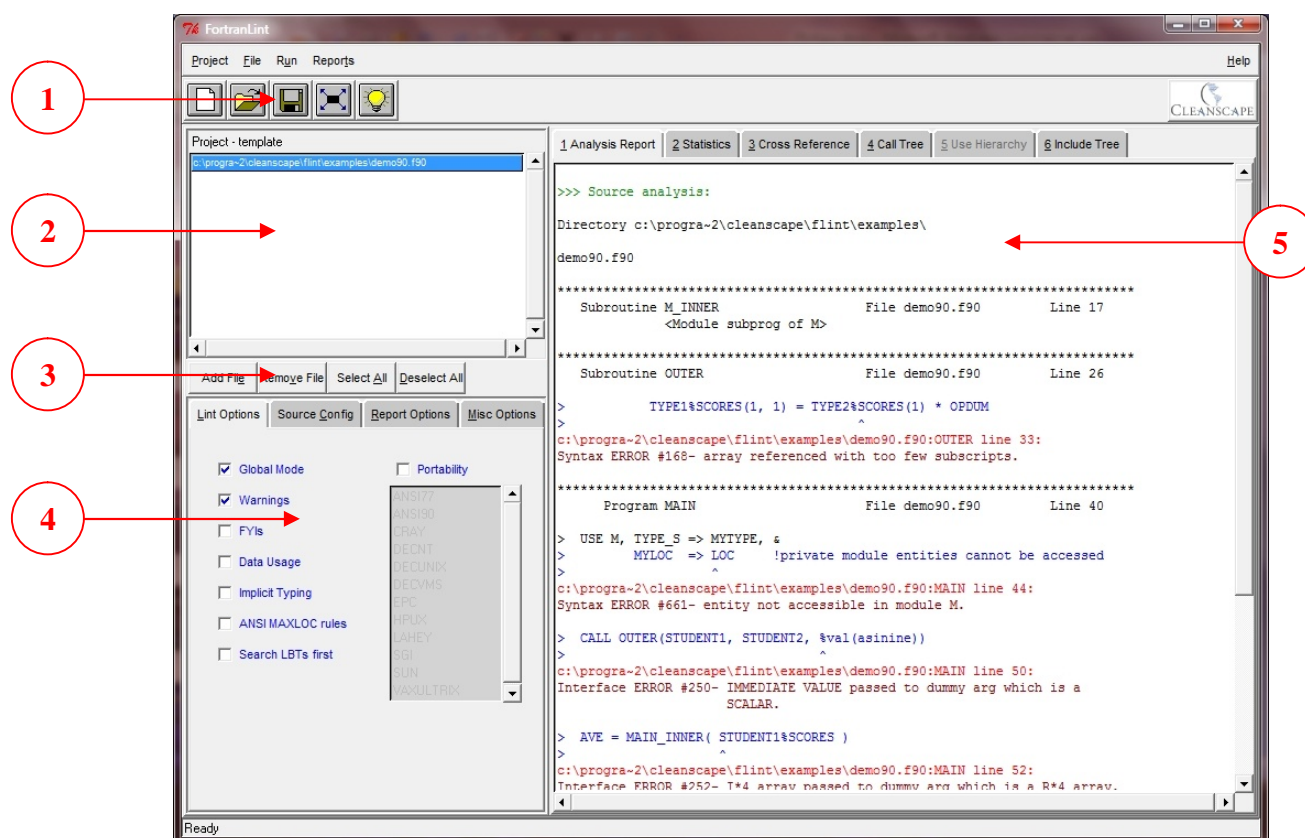
### K. Operating the GUI using the Keyboard; Keyboard Shortcuts

All aspects of the Flint GUI can be controlled from the keyboard. This capability was added to [comply](#) with the US Government's [Section 508](#) provisions.

1. Accessing dropdown menus and items using keyboard accelerators. This is the standard mode common to all Windows products.
  - a. Select the desired menu by holding down the <ALT> key, then pressing the underlined letter for that menu item. For instance, this screen image was obtained by pressing and holding <ALT>, then typing the "h" key:



- b. For information about Flint, release the <ALT> key and then press "a".
2. Navigating amongst screen elements. There are 5 screen elements in the GUI, as shown below:



- a. The <TAB> key scrolls between these five screen elements and all active items within each element. <SHIFT>+<TAB> reverses the scrolling. The item with focus will have a dotted line around its border. *Note:* Because



of the background color, the icon buttons in Element 1 will not show the dotted-line highlighting.

- b. For buttons (including radio buttons), pressing the space bar will “push” the button.
  - c. For checkboxes, pressing the space bar will “check/uncheck” the box.
  - d. For dropdown boxes, pressing the space bar will open the dropdown; the up/down arrows will navigate the dropdown, and <ENTER> key will select.
3. Keyboard shortcuts.
- a. The standard Windows shortcuts are available. For instance, pressing <F1> will bring up the Help listing; <ALT>+<F4> exits the program.
  - b. Use the alt-key combination to access a menu, then type just the underlined letter to access a submenu item. For instance, to invoke Project-Save As, one would type <ALT>+<p>, then <a>. Alternately, the arrow keys can be used to navigate submenu selections once the menu dropdown has been activated with <ALT>+<p>.
  - c. The following keyboard shortcuts are also available within the GUI:
 

|           |                                                                       |
|-----------|-----------------------------------------------------------------------|
| <ALT>+<o> | <u>O</u> pen Project                                                  |
| <ALT>+<g> | R <u>u</u> n the Analysis ( <u>G</u> o)                               |
| <ALT>+<x> | <u>E</u> xit GUI                                                      |
| <ALT>+<l> | J <u>u</u> mp to <u>L</u> int analysis tab (in Element 4)             |
| <ALT>+<c> | J <u>u</u> mp to S <u>o</u> urce <u>C</u> onfig tab (in Element 4)    |
| <ALT>+<r> | J <u>u</u> mp to <u>R</u> eports tab (in Element 4)                   |
| <ALT>+<m> | J <u>u</u> mp to <u>M</u> isc Options tab (in Element 4)              |
| <ALT>+<1> | J <u>u</u> mp to Report # <u>1</u> (Analysis report in Element 5)     |
| <ALT>+<2> | J <u>u</u> mp to Report # <u>2</u> (Statistics report in Element 5)   |
| <ALT>+<3> | J <u>u</u> mp to Report # <u>3</u> (Xref report in Element 5)         |
| <ALT>+<4> | J <u>u</u> mp to Report # <u>4</u> (Call tree report in Element 5)    |
| <ALT>+<5> | J <u>u</u> mp to Report # <u>5</u> (Include tree report in Element 5) |

#### L. Changing fonts / sizes

To change the fonts and sizes used within the GUI, edit the text file `flint.ini` located in the `main` subdirectory on Windows or your `$HOME` directory if \*nix.

In that file, you will see a section starting with `[fonts]`. Change the values from default to a value specified as follows:

*name size style*

where *name* is any font name on your system (enclose in curly braces if spaced),  
*size* is an integer font size, and  
*style* is one of: `normal` `bold` `italic` `underline`

*Example:* `report text = {Lucida Console} 9 normal`

Specifying all three font characteristics is recommended.

**NOTES:** The GUI will attempt to substitute Helvetica 10 or Courier for invalid fonts. The case of the font name can be a factor.



## PART V Running Flint from the Command Line

### A. Introduction

Flint has a command line facility suitable for standalone operation or for inclusion in scripts, e.g., for “make lint” purposes.

For details on the actual operation of Flint and its control and reporting options, refer to the companion document, [Flint Reference Manual](#).

### B. Operation

To run Flint in command line mode, you need to have set the environment variables as defined in Section 2.1.B.1.d or 2.2.B.f and registered the product as described in [Section 3](#).

The format of the Flint command line is quite simple:

```
flint [parameters] [source_filename(s)]
```

Entering `flint` without parameters yields a command summary.

Details on all the command line parameters may be found starting in Chapter 3 of the [Flint Reference Manual](#).

Other important sections in the Flint reference manual include cross-reference format/content sub-options (Chapter 8), in which very finely honed cross-reference results may be obtained, and the Unix install guide (Appendix A), which also provides details on the license daemon (not used under Windows).

**NOTE:** INCLUDE reports are available only from the Flint GUI, but they may then be saved to disk as text files; see Section 4.H.2.

### C. Return Codes

- 0 (zero) indicates that Flint ran and ran successfully without encountering any source errors.
- 1 indicates that FYI (informational) analysis messages were produced.
- 2 indicates that warning (and possibly FYI) analysis messages were produced.
- 3 indicates that error (and possibly warning and FYI) analysis messages were produced.
- 4 indicates a fatal error that prevented Fortran-lint to terminate before completion. This includes license management issues.
- If `-Mpassed` option is supplied to Flint, the return code will be set to 0 (zero) if no severe or fatal analysis messages were produced (useful in scripts), and a PASSED message like the one below will be output as well:  
`PASSED - No Flint messages above warning level generated.`

A detailed description is of course available in the analysis report. If there was a problem starting the program or securing a key, contact [support@cleanscape.net](mailto:support@cleanscape.net). If you are under maintenance, you may also contact Cleanscape Support for questions regarding any analysis output message.

For more information Flint's return codes and their uses, see Section 6.5 of the [Flint Reference Manual](#).

#### D. Command Line Example: Basic Analysis

Sample programs are located in the `examples` subdirectory. From a command prompt, run one of the following, then the “flint” command after that:

**Unix:**        `cd $FLINTHOME/../../examples`  
**Windows:** `cd %FLINTHOME%\..\examples`

`flint -g demo90.f90`

This will result in an output resembling the following. At present, Flint has 911 unique tests that comprise 1662 analyses of your Fortran code. Many of these are undetectable by a compiler because Flint is uniquely capable of analyzing the global program rather than a single file at a time. Specifically, Flint can track variable usage (e.g., reference-before-set) across call interface boundaries.

`-g` is the option to enable global mode. When starting out with Flint, we recommend you do *not* run global mode at first, but rather eliminate local errors and then tackle the global ones. This helps avoid information overload; see section 7.4 of this document.

```

FORTRAN-lint Rev 7.70 15-May-2024 13:04:29
Default options: -w -u -O207,276,76,261 --t -Xno.unreferenced.parameters
 -Xno.unused.common.variables,no.named.IEEE_*,no.named.C_*
 -Xno.named.ISO_*,no.named.COMPILER_* --x
Command options: -g

>>> Source analysis:
demo90.f90

Module M File demo90.f90 Line 4

> REAL, PRIVATE :: LOC(FOO)
> ^
demo90.f90:M line 10:
Usage WARNING #509- array subscript is not integer data type.

> REAL, PRIVATE :: LOC(FOO)
> ^
demo90.f90:M line 10:
Usage ERROR #126- local variable FOO is referenced but never set.

Subroutine M_INNER File demo90.f90 Line 17
 <Module subprog of M>

Subroutine OUTER File demo90.f90 Line 26

> TYPE1%SCORES(1, 1) = TYPE2%SCORES(1) * OPDUM
> ^
demo90.f90:OUTER line 33:
Syntax ERROR #168- array referenced with too few subscripts.

Program MAIN File demo90.f90 Line 40

> USE M, TYPE_S => MYTYPE, &
> MYLOC => LOC !private module entities cannot be accessed
> ^
demo90.f90:MAIN line 44:
Syntax ERROR #661- entity not accessible in module M.

```

```

> CALL OUTER(STUDENT1, STUDENT2, %val(asinine))
> ^
demo90.f90:MAIN line 50:
Interface ERROR #250- IMMEDIATE VALUE passed to dummy arg which is a SCALAR.

> AVE = MAIN_INNER(STUDENT1%SCORES)
> ^
demo90.f90:MAIN line 52:
Interface ERROR #252- I*4 array passed to dummy arg which is a R*4 array.

> CALL M_INNER(STUDENT1, STUDENT2)
> ^
demo90.f90:MAIN line 49:
Usage ERROR #126- local variable STUDENT2 is referenced but never set.

> CALL OUTER(STUDENT1, STUDENT2, %val(asinine))
> ^
demo90.f90:MAIN line 50:
Usage ERROR #126- local variable ASININE is referenced but never set.

> STR = GRADE(3)
> ^
demo90.f90:MAIN line 51:
Usage WARNING #127- local variable STR is set but never referenced.

Function MAIN_INNER File demo90.f90 Line 56
 <Internal subprog of MAIN>

Global checking:
Usage ERROR #742- module entity referenced but not set: M:FOO
Usage WARNING #743- module entity set but not referenced: M:AVE

```

- Default operation is controlled by file flint.cfg, located in the FLINTHOME directory. So without any operands, you get warnings (-w), data usage checking (-u), common messages omitted (-O), call tree and cross reference turned off (--t and --x, respectively), and some cross reference filters added (-X, but do not apply because the cross reference is off).
- Any of these settings can be overridden on the command line, or placed in a separate .cfg file of your choosing and specified on the command line with -E.
- Analysis results are presented in 4 lines: the “offending” line of source, then a line with a caret mark (^) to indicate the column where the issue is detected, then the source filename and line number, and finally the Flint message number with severity and text description.
- A listing of all Flint command line options is found in Chapter 3 of the [Flint Reference Manual](#).

*E. Command Line Example: Call Tree*

By turning on the call tree with -t, you get a “roadmap” of your code useful for code familiarization, whether that’s code being integrated into your project, or new programmers being integrated into your team. The added output is shown below.

```
flint -t demo90.f90
```

This is a primary tree starting at the program 'MAIN'

```
MAIN-+-M
 |
 +-M_INNER
 |
 +-OUTER--M
 |
 +-[MAIN_INNER]
```

- Tree is column-oriented, so MAIN calls M, M\_INNER, OUTER, and MAIN\_INNER. Then, OUTER calls M.
- MAIN\_INNER is in brackets because it is an internal function – a module within module M which is USED by MAIN.
- There are a number of options to call trees; these can be viewed in Chapter 7 of the [Flint Reference Manual](#).

*F. Command Line Example: Cross Reference*

By turning on the cross reference (xref) with -x, you get a listing of all symbols used in your program, including the filename, line number, and usage type (set, referenced, allocated, etc.). This is very useful for refactoring. The added output is shown below. Note the example uses -X, not -x: -X implies -x, and -Xline adds the line numbers instead of just the filename, which is the -x default. Note commands can be combined as demonstrated below: both global and xref.

```
flint -gXline demo90.f90
```

```
***** SYMBOL TABLE *****
```

```
*** Program:
```

```
MAIN : defined at line 40 of demo90.f90
 Calls- demo90.f90:M@43, demo90.f90:M::M_INNER@49,
 demo90.f90:OUTER@50,
 demo90.f90:MAIN::MAIN_INNER@52
```

```
*** Subroutines:
```

```
M_INNER : M internal : defined at line 17 to 21 of demo90.f90
 Called by- demo90.f90:MAIN@49
OUTER : defined at line 26 to 38 of demo90.f90
 Calls- demo90.f90:M@28
 Called by- demo90.f90:MAIN@50
```

```
*** Functions:
```

```
INT : I*4 : intrinsic function
 Called by- demo90.f90:MAIN::MAIN_INNER@13
```

```

KIND : I*4 : intrinsic function
 Called by- demo90.f90:MAIN::MAIN_INNER@5
MAIN_INNER : I*4 : MAIN internal : defined at line 56 to 56 of demo90.f90
 Called by- demo90.f90:MAIN@52
PRESENT : L*4 : intrinsic function
 Called by- demo90.f90:OUTER@32
SIZE : I*4 : intrinsic function
 Called by- demo90.f90:MAIN::MAIN_INNER@6

*** Modules:

M : defined at line 4 of demo90.f90
 Called by- demo90.f90:OUTER@28, demo90.f90:MAIN@43

*** Types:

MYTYPE : size = 26 bytes
 NAME : CHAR*10
 in (demo90.f90:M) is 6-D
 in (demo90.f90:M::M_INNER) is 20-R 20-S 20-R
 SCORES (2,2) : I*4 : KIND= 4
 in (demo90.f90:M) is 7-D
 in (demo90.f90:OUTER) is 33-S 33-R 35-S 35-R
TYPE_S : size = 26 bytes
 NAME : CHAR*10
 SCORES (2,2) : I*4 : KIND= 4
 in (demo90.f90:MAIN) is 52-XA

*** Records:

STUDENT1 : type TYPE_S : local
 in (demo90.f90:MAIN) is 46-D 49-XA 50-XA 52-XA
STUDENT2 : type TYPE_S : local
 in (demo90.f90:MAIN) is 46-D 49-XA 50-XA
TYPE1 : type MYTYPE : local
 in (demo90.f90:M::M_INNER) is 17-P 18-D 20-S
 in (demo90.f90:OUTER) is 26-P 29-D 33-S 35-S
TYPE2 : type MYTYPE : local
 in (demo90.f90:M::M_INNER) is 17-P 19-D 20-R 20-R
 in (demo90.f90:OUTER) is 26-P 29-D 33-R 35-R

*** Vars/Arrays:

ASININE : I*4 : local
 in (demo90.f90:MAIN) is 47-D 50-R
AVE : I*4 : public entity of module M
 in (demo90.f90:M) is 11-D
 in (demo90.f90:MAIN) is 52-S
DUM (:,:) : R*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)3-P (demo90.inc)4-D
 (demo90.inc)6-A (demo90.inc)7-A
 (demo90.inc)9-R

FOO : R*4 : public entity of module M
 in (demo90.f90:M) is 10-d 10-RB
I : I*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)6-d
 (demo90.inc)6-RSc
 (demo90.inc)9-R

J : I*4 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)7-d
 (demo90.inc)7-RSc
 (demo90.inc)9-R

LOC (adj) : R*4 : private entity of module M
 in (demo90.f90:M) is 10-D

```

```

OPDUM : I*4 : local
 in (demo90.f90:OUTER) is 26-P 30-D 32-RA 33-R 37-S
STR : CHAR*10 : local
 in (demo90.f90:MAIN) is 45-D 51-S
SUM : R*8 : KIND= 8 : local
 in (demo90.f90:MAIN::MAIN_INNER) is (demo90.inc)5-D (demo90.inc)5-I
 (demo90.inc)8-R (demo90.inc)9-R
 (demo90.inc)9-S
 (demo90.inc)10-R
 (demo90.inc)13-RA

*** Parameters:

GRADE (6) : CHAR*2
 in (demo90.f90:MAIN) is 51-R

*** Structure components:

TYPE2%NAME : CHAR*10
 in (demo90.f90:M::M_INNER) is 20-R 20-R
TYPE1%NAME : CHAR*10
 in (demo90.f90:M::M_INNER) is 20-S
TYPE1%SCORES : I*4 : KIND= 4
 in (demo90.f90:OUTER) is 33-S 35-S
TYPE2%SCORES : I*4 : KIND= 4
 in (demo90.f90:OUTER) is 33-R 35-R
STUDENT1%SCORES : I*4 : KIND= 4
 in (demo90.f90:MAIN) is 52-XA

```

- The xref is organized into different Fortran elements, such as Programs, Functions, Vars/Arrays, Parameters, and Structure Components.
- For each symbol listed, you can see you get the datatype, its scope, the filenames and line numbers it is used on, and the usage (R for Referenced, D for Defined, etc.). The list of usage symbols and their meanings is found in Chapter 8 of the [Flint Reference Manual](#) or by adding -Xlegend to your flint command line.
- Since the xref contains *all* symbols used in the program, it can get very large. Because of this, there is an extensive filtering facility. That, and other controls for the xref, can be found in Chapter 8 of the [Flint Reference Manual](#).

### G. Command Line Example: Summary Report with Refactor Data

The summary report is shown here because in addition to summary and statistics info, it offers assessments for every procedure (subroutine or function) in your program: McCabe cyclomatic complexity V(G), line count, and parameter count. All these are very useful for refactoring as well. -A enables the Assessments and implies -s, so you don't have to add -s to the command line; the Assessment section is in **red** below.

```
flint -gAV5 demo90.f90
```

```
This >>> Statistics:
```

```

Number of source files: 1

Source files: 57 lines, 1350 bytes (18% comments, 82% code)
Include files: 14 lines, 352 bytes (5% comments, 95% code)
Total parsed: 71 lines, 1702 bytes (15% comments, 85% code)

```



```

Total subprograms: 5
 Subroutines: 2
 Functions: 1
 Program: 1
 Block Data: 0
 Modules: 1

```

#### Program Unit Assessments

```

PROCEDURE NAME TYPE LSTART LINECT PARMCT v(G)
demo90.f90::M_INNER (SUB) 17 5 2 2
demo90.f90::OUTER (SUB) 26 13 3 2
demo90.f90::MAIN (PRG) 40 15 0 1
demo90.f90::MAIN_INNER (FCN) 56 1 1 5

```

#### Individual message summary

```

Usage ERR #126- 3x: local * * is referenced but never set.
Usage WARN #127- 1x: local variable * is set but never referenced.
Syntax ERR #168- 1x: array referenced with too few subscripts.
Intrfc ERR #250- 1x: * passed to dummy arg which is * *.
Intrfc ERR #252- 1x: * array passed to dummy arg which is a * array.
Usage WARN #509- 1x: array subscript is not integer data type.
Syntax ERR #661- 1x: entity not accessible in module *.
Usage ERR #742- 1x: module entity referenced but not set: *, *
Usage WARN #743- 1x: module entity set but not referenced: *, *

```

Total messages: 11

|                   | Errors | Warnings | FYIs   |
|-------------------|--------|----------|--------|
| Syntax:           | 2      | 0        | <supp> |
| Global Interface: | 2      | 0        | <supp> |
| Data usage:       | 4      | 3        | <supp> |
| Implicit typing:  |        | <supp>   |        |

## H. Other useful command line options and external utilities

A summary of Flint command line options is available by typing just `flint` at the command line. Sub-options can be viewed for these primary options by typing one of: `-Fhelp` `-Mhelp` `-Xhelp` `-Thelp` after the `flint` command.

`-S <fname>` outputs the various reports to the filename given by `<fname>`, with the extensions `.lnt` for lint analysis, `.tre` for call tree, `.xrf` for cross reference, and `.stt` for statistics report. These are plain text files.

`-e` specifies 132-character source lines

`-p` Run the preprocessor (`-#` allows you to specify the preprocessor). Flint obeys Fortran convention to run preprocessor on any files whose extensions are capitalized - `.F90`, `.FOR`, etc.

`-I <path>` defines include directory to search for include files specified with preprocessor `#include` or Fortran `INCLUDE` statement. The directory containing the sourcefile, and the current directory Flint is being run from, are searched by default and in that order.

`-D <macro>` Add macros to be used by `#define` preprocessor statements in your sourcefiles.

**-M Miscellaneous options. In particular, -Mpassed will set return code to zero and generate a PASSED message if no messages of error level severity were found in the code.**

**filtflint.** Analysis results of a subset of the sourcebase can be viewed within the context of the entire program using external program `filtflint`. This is useful for a large, stable sourcebase where perhaps 1-2 sourcefiles are being modified – the analysis can generate megabytes of data when only a fraction of that total is relevant to the sourcefiles of interest. See Section 7.5 for more details.

**usescan.** If you use modules, and wish to analyze one file (or a subset of files) in your project, you must also supply the files containing the USED modules. Cleanscape helps automate this process: you supply the list of all files comprising your program, then have external program `usescan` construct `USE` rules for Flint's reference during analysis. See Section 7.3 for more details.

## PART VI Running Flint from IDEs

### A. Overview

For customers who develop their code in Integrated Development Environments (IDEs), it is more useful to operate Flint from within the environment.

Advantages include:

- Flint obtains project information, including the file list, include directories, defines/undefines, and excluded files – no need to re-specify.
- Deep integration: When applicable, the equivalent Flint command is derived from controls set within the project. For instance, setting Intel Visual Fortran's "Fixed Form Line Length" to 132 will enable Flint's `-e` option.
- Flint uses the output window of the IDE for results and, if available in the IDE, links to the source line using the IDE's internal editor.
- Fewer windows to shuffle between.
- **NOTE:** The number of tools varies between IDEs; see the Tools menu in your IDE for availability. Section C below is a full description of all tools.

**NEW v7.4** • When Project Setup is run, a GUI `.csi` control file is also created, meaning you can run Flint from the GUI (thus obtaining call tree and cross reference reports with hyperlinks from the GUI), while still having the option to run Flint's analysis and get results strictly within the IDE!

Current IDEs supported:

- ✓ *Microsoft Visual Studio versions: VS6 → VS2019*
- ✓ Other IDEs may be supported via the External Editor option in the Cleanscape GUI (see Sec. 4.B.6).

**NEW v7.6** The installation program will scan your system for later Visual Studio version(s) on your machine, and automatically install the Cleanscape Tools to each VS found.

Eclipse integration is available. To obtain integration instructions or request deep integration for your IDE, email [sales@cleanscape.net](mailto:sales@cleanscape.net).

### B. Installation

To run Flint from IDEs, you need to have registered and activated the product as described in Section 3.

Integration with existing IDEs occurs automatically when you install Flint. If you added a new version or an altogether new IDE, use the batch file corresponding to the new IDE found in the 'ideinstall' subdirectory, or contact [support@cleanscape.net](mailto:support@cleanscape.net) for assistance.

**NOTE:** Care has been taken to allow users who are not logged in as "owner" to successfully install the Visual Studio tools; if you encounter problems, install as owner (or have your Administrator install the product for you) and notify [support@cleanscape.net](mailto:support@cleanscape.net).

### C. Operation

The installer will place four “External Tools”, found in the “Tools” dropdown menu for your IDE.

In the titles for each tool, below as well as on the “Tools” dropdown menu, the underlined letter indicates the default “accelerator key” for that tool. Clicking ALT-T then the underlined accelerator key represents a shortcut to running that tool. For details on the actual operation of Fortran-lint and its control and reporting options, refer to the companion document, *flintman.pdf*, located in the ‘doc’ subdirectory.

#### 1. Flint Project Setup - tool by Cleanscape

**NOTE:** This option must be run first to successfully analyze projects (tool #3).

This option does not run any analyses, but instead parses the IDE project file for the list of source files, include directories, defines/undefines, and other settings comprising the current project. These values are then placed in a Flint `<IDEproject>.cfg` file, created and maintained by this Setup tool.

Rerun Setup each time the project changes (changes to the source file list, include directories, or defines).

**HINT:** Be sure to save your project each time before running Setup!

Because the `<IDEproject>.cfg` file will be rewritten each time the tool is run, do not edit the file! Instead, use file `std.cfg` to specify analysis control settings to Flint. `std.cfg` is never changed automatically, and is therefore the place to specify Flint options on a per-project basis.

**NEW v7.4** A third file, `<IDEproject>.csi`, is also created. This file translates both `.cfg` files into a control file for the Flint GUI. In so doing, a complete view of your project is *automatically* available to the GUI, meaning that other reports more readily viewed and browsed (via hyperlinking) from the GUI, such as call tree and cross reference, are easily obtained.

Like the project file, this file will also be rewritten each Project Setup, so do not edit it! Instead, save any customizations to your `std.cfg` file. For assistance, contact [support@cleanscape.net](mailto:support@cleanscape.net). Just in case, a backup of the existing `.csi` file is made before a rewrite occurs.

All these files are stored in the Visual Studio project directory.

Rerun this tool each time the project contents change (e.g., when files are added or deleted). Again, the contents of any `std.cfg` file are preserved.

#### 2. Flint Single File - tool by Cleanscape

Use this tool to run Flint over a single file in your project. Analysis options are read from file `std.cfg` located in the Visual Studio project directory.

Analysis results appear in the Output window. When available in your IDE, double-clicking on any error message will cause the IDE’s built-in editor to jump to the line number in the source file related to the analysis message.

The system include directories for the IDE you're running are obtained automatically for each analysis – great if you have multiple versions installed or you don't have the INCLUDE environment variable set.

### 3. Flint Entire Project - tool by Cleanscape

Use this tool (after having set up the project by using tool #1 above) to run Flint over the *entire* project sourcebase. As with the single module tool (#2 above), analysis results appear in the Output window; when available in your IDE, double-clicking on any error message will cause the IDE's built-in editor to jump to the line number in the source file related to the analysis message.

The system include directories for the version of Visual Studio you're running are obtained automatically for each analysis – great if you have multiple versions installed or you don't have the INCLUDE environment variable set.

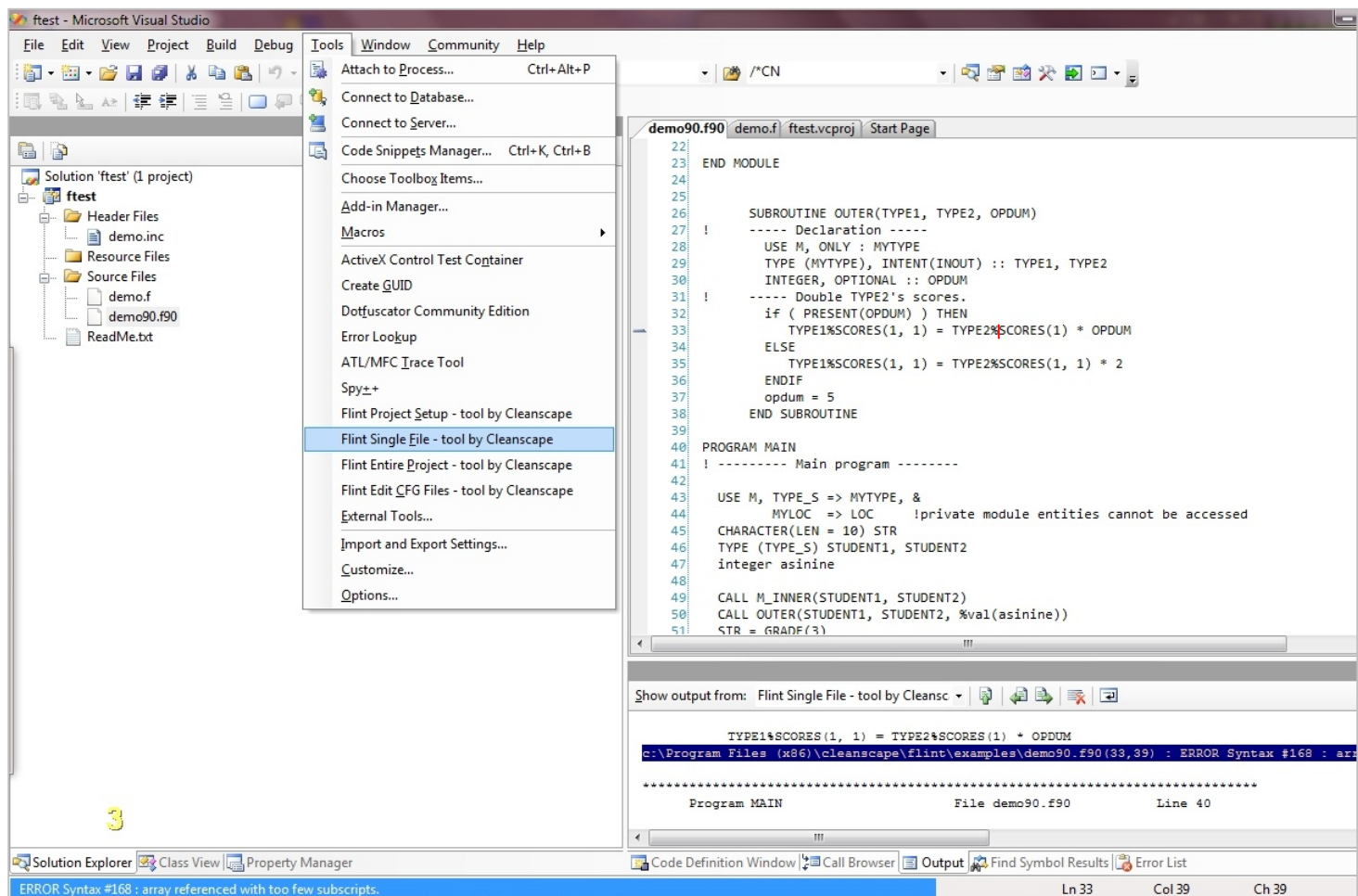
### 4. Flint Edit CFG Files - tool by Cleanscape

This tool conveniently invokes Notepad to edit the `std.cfg` and the `<IDEproject>.cfg` files (described in #2 above) associated with the current project. Do not edit the `<IDEproject>.cfg` file as it can be overwritten at any time; it is opened here for quick reference only. Place any additions or overrides into your `std.cfg` file.

Remember, it is possible to run Flint analysis from the GUI after running Project Setup (tool #1) above. By so doing, you can take advantage of other reports, like the cross reference and call tree, which are not as readily viewed in the IDE's output window. Plus, such reports are hyperlinked in the GUI, meaning you can click on any entry and the relevant line of source is presented in your favorite code editor! At present, Visual Studio's internal editor is opened when you click on a link in the GUI after using Project Setup; another editor can be chosen within the GUI, and any such choice is preserved later.

The screenshot below shows the Cleanscape tools installed in Visual Studio's Tools dropdown menu. It also shows the analysis results of the current project.

The highlighted ( **blue background** ) line in the Output window (bottom) was double clicked, resulting in Visual Studio's editor (top window) jumping to the source file / source line (line 33) that caused the analysis message.



#### D. Uninstallation

Manually remove any tool(s) from Visual Studio:

0. Open the Visual Studio IDE.

1. From the Tools dropdown menu, select "External Tools..." (in VS 6 select "Customize..." then click on the "Tools" tab in the resulting dialog).
2. Click the tool you wish to delete.
3. Click the "Delete" button in the right side of the dialog box (in VS6 click on the red 'X' in the upper right corner).

## PART VII Miscellaneous Information

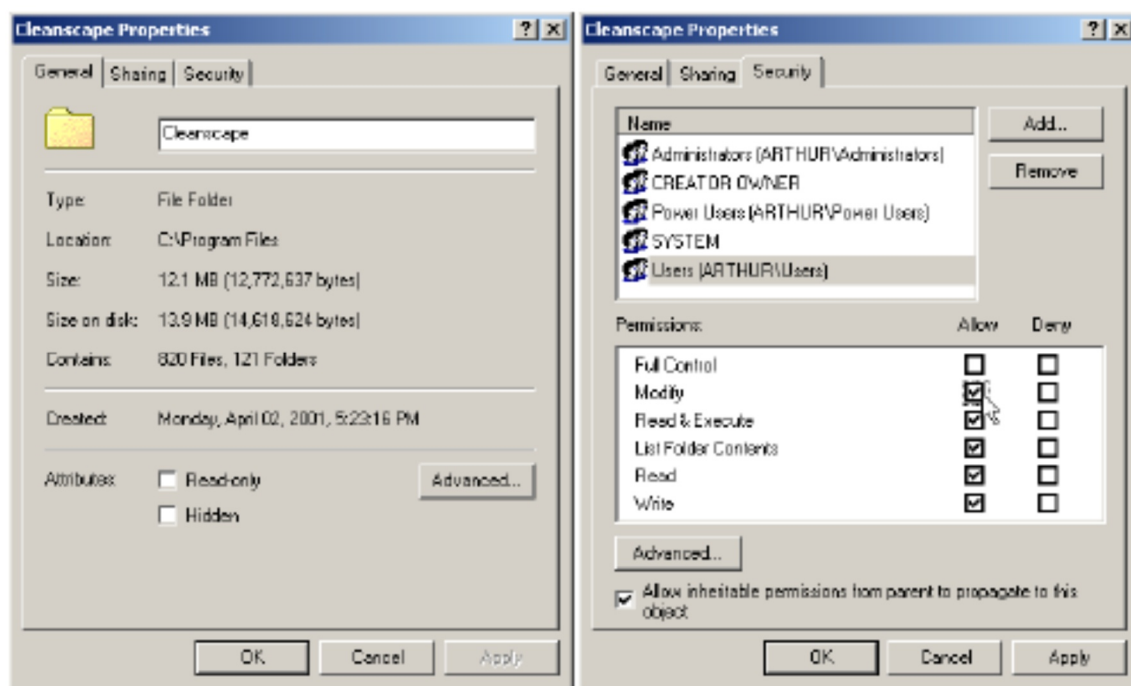
### 7.1 ADDITIONAL STEPS FOR WINDOWS PERMISSIONS

#### A. Applicability

1. This section applies to users running Windows 2000+ who belong to the “Users” group, and only to that group.

#### B. Details

1. For Flint to run correctly, users must have “write” and “modify” access rights to the installation directory and all its subdirectories.
  - a. Log in as “administrator” and finish installing Flint.
  - b. Double-click on the “My Computer” icon on the desktop.
  - c. Navigate to and double-click on the installation folder. Select Properties from the sub-menu.
  - d. Select “Security” tab on the Properties screen:



- e. Select the “Users” group and enable “Modify” and “Write” permissions.
- f. Click the “Apply” button.
- g. Click the “OK” button. This should close the Properties window.
- h. Flint is now ready to run on Win2k for the “Users” group.





## 7.2 ADDING AN EXTERNAL EDITOR TO THE GUI USING *SETEditor*

### A. Introduction

By popular demand, Cleanscape has added the ability for users to specify their own favorite editor to any Cleanscape product (as opposed to submitting a feature request to Cleanscape Support). This is implemented via an external program called *seteditor*, located in the 'bin' subdirectory.

User contributions welcome! Send them to [support@cleanscape.net](mailto:support@cleanscape.net); any contributions will receive appropriate credit and be placed in a "master" file located at [http://www.cleanscape.net/products/contributed\\_editors.html](http://www.cleanscape.net/products/contributed_editors.html).

### B. Operation

On any platform, it is possible to edit file *myeditor.lst* manually; see the comments inside the file, which is located in *bin* subdirectory on Windows or your *\$HOME* directory on Unix/Linux. The Unix/Linux session on the next page shows the contents of *myeditor.lst* (which looks substantially similar under Windows).

#### Windows.

You can either run *seteditor* from the command line or via Explorer.

From a DOS shell (cmd or command prompt), run the following command:

```
"<install_dir>\bin\seteditor"
```

From Explorer, navigate to the above directory and then double-click *seteditor.exe*.

#### Unix.

From a shell prompt, run the following command:

```
<install_dir>/bin/seteditor
```

Three pop-up dialogs (Windows) or a sequence of shell interactions (Unix/Linux) will guide you through

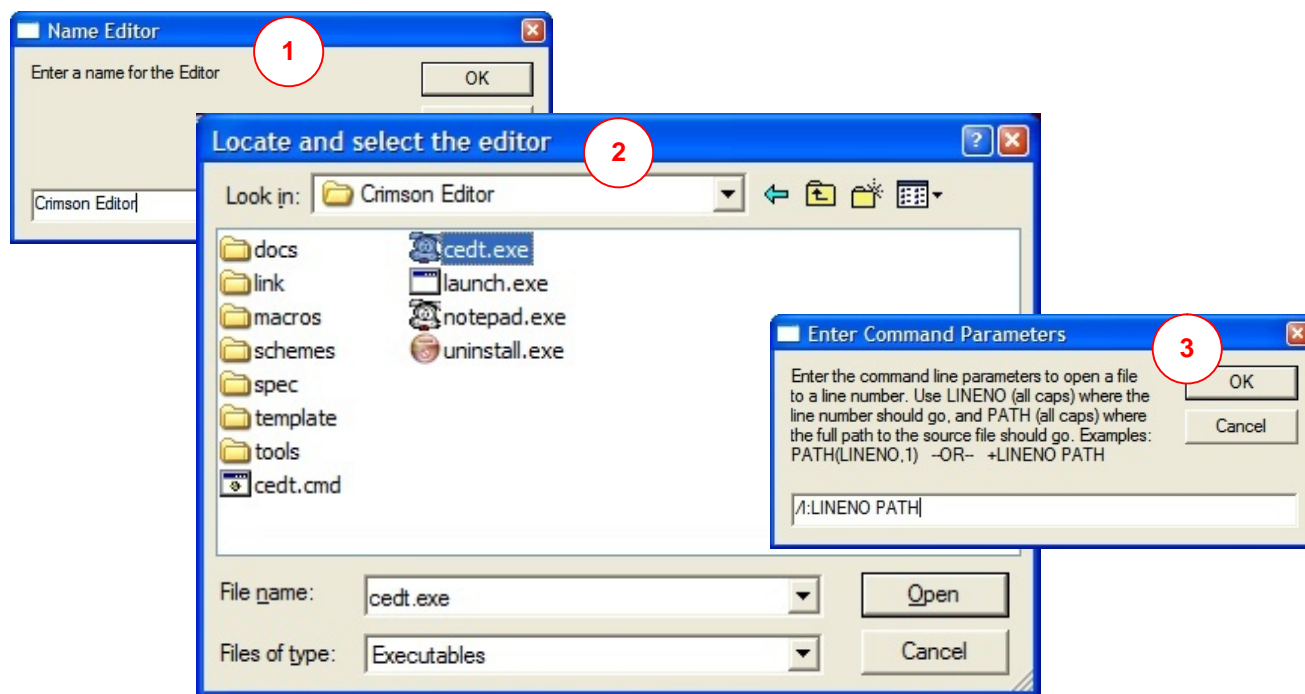
1. Naming the editor (a label identifier)
2. Locating the editor executable itself
3. Setting command line parameters to open a file and jump to a line number.

A sample Windows session depicting the dialogs for all three steps (and labeled as such) is shown on the next page, as is a Unix/Linux shell session.

**NOTE:** Refer to your editor's documentation to get the editor's command line information required (i.e., specifying the filename to open and the line number to jump to when opening the file). If your editor does not support jumping to line numbers from the command line, you can still invoke the editor but it will be impossible to align the analysis message to the "offending" source line.

We also recommend setting your editor to run as a single instance.

Any number of editors may be added in this fashion. Added file information is stored in file *myeditor.lst*; once successfully added, email your *myeditor.lst* file to [support@cleanscape.net](mailto:support@cleanscape.net) for inclusion in a Master file to share with other Cleanscape customers!



```
suse:/home/chris
suse:~$ /usr/local/cleanscape/bin/seteditor

This program adds an external editor to the Cleanscape GUI(s).
You will need to supply the command line switches for loading a file and
jumping to a line number. Enter 'quit' to consult the editor documentation
first if necessary, or <Enter> to proceed:

Use CTRL-C to exit at any of the following prompts.
Enter a name for the Editor: Kwrite
Enter the path for the Editor (default /usr/bin): /opt/kde3/bin
Enter the filename for the Editor (default kwrite):
Is this a text-based editor intended to run inside a console window? (y/n): n

Enter the command line parameters to open a file to a line number.
Use LINENO (all caps) where the line number should go, and
PATH (all caps) where the full path to the source file should go.
Examples: PATH(LINENO,1) --OR-- +LINENO PATH
Parameters (default +LINENO PATH): --line LINENO PATH
Kwrite has been added to the list for Cleanscape GUI(s).
suse:~$ cat myeditor.lst
This file holds information required to add an editor to the Cleanscape GUI.
A line with '#' in column one is a comment.

Program "seteditor" interactively adds a file, or edit this file using the
template/example below (sans '#' in column one). "path_line" in the template
represents your editor's command line parameters for specifying
1) the source file's fully qualified pathname (denoted as PATH) and
2) how to jump to a specified line when opening a file (denoted as LINENO).

Note that PATH and LINENO must be in all caps, the executable starts with
'/', and the editor path does NOT have a trailing '/'.

"text_based" in the template is either a Y or a N and indicates whether the
editor is text-based and intended to run inside a console window. This
field is ignored (but must still be present) for windows.

TEMPLATE:
editor-label__editor-filename__editor-path__text-based__path-line

EXAMPLE:
Joe__/joe__usr/bin__Y__+LINENO PATH

Kwrite__kwrite__opt/kde3/bin__N__--line LINENO PATH
suse:~$
```

### 7.3 FIND MISSING MODULES WITH `USESCAN`

#### *Introduction*

If you use modules, and wish to test one file (or a subset of files) in your project, you need a way to tell Flint where those modules are located. (If any are missing, you get error message #731, and missing modules can lead to *many* extra error messages.)

You may have had a similar experience with your compiler: you have to manually add sourcefiles containing modules, and order them correctly so they compile. However, unlike a compiler, Cleanscape helps automate this process: all you need do is supply the list of files comprising your program (*not* `#included` or `INCLUDED` files), then have external program `usescan` construct `USE` rules for Flint's reference during analysis.

And you can use it for your builds too! Dependencies and ordering can be obtained with `usescan`.

`usescan` is located in `$CSIAPPPBASE/bin` on \*nix and `$FLINTHOME\..\bin` on Windows.

NOTE: *you do not need to create projects unless you use modules and those modules are in different files!*

There are two modes to `usescan` operation: one that generates `USE` dependencies, the other that looks up what files need to be on the sourcefile list for Flint to run without `USE` issues. Each mode is discussed in the next sections.

#### *Generate `USE` dependencies*

When you build `USE` information for your project, you will need to know where your sourcefiles and include directories are located on your disk, plus any define macros which control the flow of the Fortran program's execution. There is a 'pause' at Step 2 in the program to allow you to gather this information, or <CTRL-C> to stop and perform it at a later time when you do have the information.

On the next page is a sample `usescan` session. User lines of input are colored **green** as shown. Follow the online prompts. They are quite verbose, so please follow carefully. We suggest you start in the source directory to minimize typing.

Once done, there will be two files in `$HOME/.flint/projects` (\*nix) or `%FLINTHOME%\..\projects` (Windows) whose base names are what you provided during this session. Neither is intended to be edited by hand.

Rerunning `usescan` *only* needs to be performed if/when the project changes (new or deleted files, new include directories, added define macros).

```
~/steve/dev/fortran/project1 $ SCSIAPPBASE/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 environment as closely as possible, and provide contextual information when only a portion of the entire sourcebase is being tested. There are 5 steps.

STEP 1: provide a single-word name for the project - no spaces, no extension...

```
fortproj1
```

Project file name is /home/steve/.flint/projects/fortproj1.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'.

```
main.F90
```

File(s) accepted. Enter next file, or 'done': **list\*.f90**

File(s) accepted. Enter next file, or 'done': **done**

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

```
done
```

usescan will rely on file extension to determine free/fixed source form.

STEP 3a) Do you wish to override this approach? (y/n) **n**

STEP 3b) Are any sourcelines >72 columns? (y/n) **y**

STEP 4: enter include or #include paths. No need to relist source directories. When done, enter 'done'.

Enter include or #include directory path (or 'done'):

```
./include
```

Accepted.

Enter next include or #include directory path (or 'done'):

```
done
```

STEP 5: 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
```

Building project and dependency data...

Done! Project file is - /home/steve/.flint/projects/fortproj1.cfp

Project file generation complete. If the project changes (e.g., files added), rerun casaf -G; if only sourcefile content has changed, try casaf -U to update instead.

*Obtain USE dependency list*

Once you have generated USE dependencies for your project, you can now obtain a list to be included on the Flint command line, or placed in a config file (which can be specified by using `-E` on the Flint command line).

The format for this mode is

```
usescan -g -p <projname> file1 [file2, file3,...]
```

Here is an example based on the above run. Note the output is actually one line to stdout; it appears broken here because of automatic word wrap:

```
~/tests $ $CSIAAPPBASE/bin/usescan -g -p fortproj1 main.F

-I/home/steve/fortran/project1/include -I/home/steve/fortran/project1
/home/steve/fortran/project1/global_sum_module.f
/home/steve/fortran/project1/comm_module.f
/home/steve/fortran/project1/util_module.f
/home/steve/fortran/project1/term_module.f
/home/steve/fortran/project1/init_module.f
/home/steve/fortran/project1/modules.f /home/steve/fortran/project1/utilio.f
/home/steve/fortran/project1/EMIS_VARS.F
/home/steve/fortran/project1/BIOG_EMIS.F /home/steve/fortran/project1/main.F
```

You could either copy/paste the output to a Flint command, or use redirection '`>`' to send the output to a Flint config file, then add the file to the Flint command after `-E`:

```
~/tests $ $CSIAAPPBASE/bin/usescan -g -p fortproj1 main.F > testproj1.cfg
~/tests $ flint -E testproj1.cfg

FORTRAN-lint Rev 7.55 16-Oct-2020 12:32:40
...
```

*usescan project creation alternatives*

`-U` updates a usescan project file only for file checksums and inter-file dependencies. You will be asked for the name of the project to update. If new files have been added, a full project regeneration using `usescan -G` will be required.

`-INH` is a special mode to inherit an existing project and then build on it to create a new project. This is very useful when there are sub-elements to your project (e.g., libraries) and the overall program is built upon these sub-elements. You will be asked for the name of the project to inherit from, then the interactive portion will begin so you can enter additional sourcefiles, include directories and preprocessor defines.





## 7.4 MINIMIZING FALSE POSITIVES FOR ENTIRE PROJECTS

### Introduction

Remember the oft-used GIGO principle – Garbage In, Garbage Out.

When analyzing an entire project, we first need to ensure that all the files comprising the project (and *only* those files) have been specified, ensure Flint has the source layout (72- vs. 132-columns), and that the statements are in proper order (some compilers allow alternative program layouts). Without these relatively minor safeguards, error messages unrelated to actual code quality can quickly pile up.

**USAGE NOTE:** All steps are available through either the GUI or command line.

### Step 0: gather the file list for the entire project

This list can be obtained from makefiles. It's also possible to extract the sourcefile list automatically from .vfproj files if using Visual Studio; see section 6.C above regarding the Flint Project Setup tool. Using either of these approaches helps ensure the file list that Flint processes will match that of the compiler.

If you don't have makefiles nor use Visual Studio, for most projects you can select the source files from a directory tree. It's important it's *only* sourcefiles, *not* include files, data files, or other artifacts. Sourcefile extensions for Fortran are most commonly { f F f77 F77 f90 F90 f95 F95 f03 F03 f08 F08 f15 F15 for FOR }.

In the GUI, you can navigate the various folders and build a sourcefile list as described in Section 4.F. Once your list of files is complete, be sure to save the project as described in Section 4.E!

For command line operation, it's usually easiest to use the OS to dump the filelist into a single file, then use that file as input to Flint. For example (Windows):

```
dir /s /b c:\sam\proj1*.f90 >> c:\sam\flint\proj1.cfg
dir /s /b c:\sam\proj1*.f >> c:\sam\flint\proj1.cfg
```

Using /s with /b prints the full pathname to each sourcefile, one per line. All files matching the criteria are then stored in proj1.cfg, which will be input to Flint using the -E command line option (example in the next step).

### Step 1: check file list integrity

Missing files are a leading cause of false positives; in this step, we run a very limited test to confirm there are no missing USE modules or INCLUDE files. Flint will also check the line length of your sourcecode, and whether any statements are out of order (statement functions can be especially troublesome).

GUI: Select "Check Filelist Integrity" on the Lint Options tab, then click the Run button.

Command line: Cleanscape has preconfigured the options and stored them in file %FLINTHOME%\prepl.cfg, or you easily add them yourself: use either

```
flint -E c:\sam\flint\proj1.cfg -E %FLINTHOME%\prepl.cfg —OR—
flint -Oall,+731,+1,+74 -E c:\sam\flint\proj1.cfg
```

If you get a terminating error message regarding columns exceeding 72 characters, rerun after selecting "132 Columns" on the Source Config tab (GUI)

or with `-e` (command line), or rarely (for code using sequence numbers) by adding `-Mstrict72` to the command line or your default `flint.cfg` file instead of `-e`.

Correct any errors reported – these will all be severe and will need resolution before proceeding. This will mostly consist of adding files to the project; if you need assistance, contact [support@cleanscape.net](mailto:support@cleanscape.net).

### *Step 2: check for missing procedures and unsupported Fortran*

A second set of analyses will determine whether there are missing functions/subroutines. Missing procedures can result in numerous error messages regarding, for instance, “reference before set”, so for errors reported during this step, it’s vital to find and add the files containing the missing procedures to the project filelist.

This check will also verify that Flint can process your code correctly; most notably, many aspects of object-oriented Fortran are not supported at this time.

**GUI:** Select “Check Missing Procedures” on the Lint Options tab, then click the Run button.

**Command line:** These settings are stored in `%FLINTHOME%\prep2.cfg`:

```
flint -E c:\sam\flint\proj1.cfg -E %FLINTHOME%\prep2.cfg —OR—
flint -gOall,+129,+130,+28 -E c:\sam\flint\proj1.cfg
```

Unsupported Fortran syntax will be flagged as #28. If you receive any of these messages, stop here and contact [support@cleanscape.net](mailto:support@cleanscape.net).

Missing procedures will be reported in a group as part of messages 129 or 130. If there is no source code for some procedures (e.g., they are library routines), it’s fast and easy to load or even create “stub” procedures that provide not only the missing procedure by name, but also provide information about all of its arguments. Cleanscape has a number of pre-written stubs in `$FLINTHOME/*.lsh`.

- Flint automatically references stubs for system calls that can be called directly from Fortran, such as `copysign`, `fgetc`, or `flush`.
- Flint also automatically references stubs for F03 IEEE and ISO intrinsics.
- If you use MPI or NetCDF, you can add Cleanscape-provided stubs for their associated procedures by including `MPI.lsh` or `NetCDF.lsh` in your file list.
- You can easily write stubs yourself and include the stub file in your analysis! Refer at any of the `.lsh` files in `$FLINTHOME` for reference. *Example:* `NCTLEN` in file `NetCDF.lsh`:

```
integer function NCTLEN(TYPE/r, RCODE/s)
integer, intent (in) :: TYPE
integer, intent (out) :: RCODE = 0
NCTLEN = 0
end function NCTLEN
```

As you can see, the stub has directives associated with each argument: `/r` informs Flint that `TYPE` is referenced, and `/s` informs Flint that `RCODE` is set. By specifying `INTENT` in the stub, Flint is able to inform you if a variable is being used incorrectly (e.g., an element of an unallocated array is the actual argument for `RCODE`). Finally, the stub sets a return value for the function so Flint does not report that the function does not return one.

There are a number of such directives, and they can be combined; refer to Chapter 9 of the [Flint Reference Manual](#) for more details. Cleanscape can also create stub files for you; contact [sales@cleanscape.net](mailto:sales@cleanscape.net) for further information.

Once created, simply add the stub (.lsh) file to your analysis file list.

- Cleanscape can create stubs for you! As part of your purchase, you are eligible for up to 50 arguments **FREE**. Additional arguments are available at a very low rate (currently US\$15 per argument). Price subject to change without notice; contact [sales@cleanscape.net](mailto:sales@cleanscape.net) for further information.

(A function's return or RESULT is counted as one argument, so writing the stub for `integer function foo(a, b, c)` would count as four arguments.)

### *Step 3: run a full analysis – in stages*

If you've never run your code through a static analyzer before, you may be surprised (and maybe overwhelmed) by the quantity of messages you encounter. Also, it's possible that errors incurred along the way generate additional messages that would go away once the first, underlying issue is resolved.

Therefore, we recommend that one builds to a full analysis in stages. The following table lists out options and when to run them. For completeness, it also includes the pre-analysis discussed in detail in Steps 1 and 2 above.

**USAGE NOTE – GUI:** Deselect “Check Filelist Integrity” and “Check Missing Procedures” on the Lint Options tab for any stages after Pre-1 or Pre-2. This is critical, since these two settings disable any other analysis options specified by you in the GUI.

| RECOMMENDED FLINT ANALYSIS STAGES FOR LARGE PROJECTS                                                       |                             |                                                    |                                               |                                                                                                     |
|------------------------------------------------------------------------------------------------------------|-----------------------------|----------------------------------------------------|-----------------------------------------------|-----------------------------------------------------------------------------------------------------|
| Stage                                                                                                      | Description                 | GUI Checkbox Options                               | Command Line Options                          | Comments/Recommendations                                                                            |
| Pre-1                                                                                                      | Project Pre-Analysis step 1 | Check Filelist Integrity                           | -Oall,+731,+1,+74                             |                                                                                                     |
| Pre-2                                                                                                      | Project Pre-Analysis step 2 | Check Missing Procedures                           | -gOall,+129,+130,+28                          |                                                                                                     |
| 1                                                                                                          | Errors only                 | [Deselect all checkboxes]                          | --w --u                                       | Turn xref, call tree off                                                                            |
| 2                                                                                                          | Errors+Warnings             | Warnings                                           | [default]                                     |                                                                                                     |
| 3                                                                                                          | Errors+Warnings+FYL         | Warnings, FYL                                      | --f                                           |                                                                                                     |
| 4                                                                                                          | Global mode, Errors only    | Global Mode<br>[All other checkboxes off]          | --w -g                                        | Turn xref, call tree off<br>Leave data usage on                                                     |
| 5                                                                                                          | Global mode + any desired   | Global Mode, ...                                   | -g ...                                        |                                                                                                     |
| Can occur at any Stage after Stage 1                                                                       |                             |                                                    |                                               |                                                                                                     |
|                                                                                                            | Implicit Data Typing        | Implicit Typing                                    | -m                                            |                                                                                                     |
|                                                                                                            | Missing #ifdef wrappers     | empty "Define Symbols" box<br>on Misc Options tab  | no -D on command line                         | Watch for message #126 or #742:<br>ref never set                                                    |
|                                                                                                            | Include/#include Report     | Include Tree                                       | N/A                                           | Help diagnose pernicious errors                                                                     |
|                                                                                                            | Procedure Assessment v(G)   | Assess suboption to Statistics                     | -A<sort>                                      | Appears in Statistics Report                                                                        |
|                                                                                                            | Cross Reference             | Cross-Reference                                    | -x   -X<option>                               | Can be large, time-consuming;<br>see Sec. 8.4 in Flint Reference<br>Manual to filter/manage results |
| Can occur after Stage 5                                                                                    |                             |                                                    |                                               |                                                                                                     |
| 6                                                                                                          | Dataflow (advanced)         | Data-flow analysis<br>Data-flow analysis w/ Global | -Fon <single_file ><br>-gFon <full_filelist > | Expert use only!<br>Not every run - resource intensive                                              |
| Questions? Email <a href="mailto:support@cleanscape.net">support@cleanscape.net</a> or phone +706-245-1070 |                             |                                                    |                                               |                                                                                                     |

**Congratulations – you have a complete Flint run!**

## 7.5 TESTING SELECTED FILES IN PROGRAM CONTEXT USING *FILTFLINT*

**USAGE NOTE – GUI:** This process has been implemented in the Flint GUI. Continue reading here for a more thorough understanding of filtering, then see Section 4.5(c) to use in the GUI.

### A. Introduction

Also by popular demand, Cleanscape has added the ability for users to view the results of 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. An analysis of the entire sourcebase can generate megabytes of output data when only a fraction of that total is relevant to the sourcefiles of interest (hereinafter referred to as UUTs, for Units Under Test).

A few years ago, considerable effort was made to create FDB (Fortran DataBase) files, which contained global analysis information. Once the FDB was created, it would be possible to analyze just a few files and use this pre-existing data, analogous to incremental linking – only the specified files would require re-analysis. This functionality still exists, and you can read about it in chapter 9 of the [Flint Reference Manual](#), and may still be useful for extremely large projects whose analysis exceeds a few minutes.

Over time, however, processors have evolved to the extent that analysis runs that took hours a decade ago now take minutes or even seconds. The FDB process takes effort for users to maintain, so with version 7.5 of Flint, we now recommend this alternative method: filtering with *filtflint*.

Users can filter the results of an analysis of the entire sourcebase for select UUTs. In so doing, *just* the analysis results, call tree, and cross reference info relevant to those UUTs are presented to the user – probably a tiny fraction of the amount produced otherwise! (By the way, the reduction in cross-reference and call tree results would not occur by using the older FDB method.)

### B. Filter results from an existing Flint run

This option is useful for very large projects that take a long time to run a complete Flint analysis: *filtflint* may be run again and again over the same set of results, perhaps changing the files of interest (UUTs). Of course, Flint needs to be rerun each time a sourcefile is changed.

Review and apply the steps described in detail in Section 7.3 just above. This ensures relevant results while minimizing false positives.

Run *flint* from the command line, along with any options you desire. For a complete list of Flint options, see Section 3 of *flintman.pdf*, located in your 'doc' subdirectory.

**NOTE:** if you want a cross reference report, be sure to add *-xline*; if you want a call tree, add *-xline* and *-t*. Use *-s*, *-+*, or the OS redirection operator *'>'* to create a *.lnt* results file (or set of results files, if *-s* or *-+* is used). Then run

```
filtflint -L <.lnt_results_file> < -U uut1 [, uut2, ...] >
```

If you would like to split the results into separate analysis, cross reference, call tree, and statistics reports, add `-S` to the `filtflint` command line. `filtflint` will automatically do this if `.lnt_results_file` is itself part of a set of results files.

`filtflint` will generate up to four results files ending with `.lnt.filt`, `.xrf.filt`, `.tre.filt`, and `.stt.filt`. The file stem will be the same as `.lnt_results_file` specified with `-L`, or you can add `-O` along with a new file stem.

### C. Run an analysis and immediately filter those results

You can both generate a new Flint analysis and filter those results using the `-A` option for `filtflint`. This is an “all-in-one” option where Flint is automatically invoked, and once Flint analysis is complete, filtering automatically takes place.

Review and apply the steps described in detail in Section 7.3 just above. This ensures relevant results while minimizing false positives.

```
filtflint -A <config_file> <-U uut1 [, uut2, ...] > [-S] [-O filestem]
```

`config_file` is a Flint configuration (`.cfg`) file. You can read more about config files in Section 3.3 of `flintman.pdf`, located in your ‘doc’ subdirectory.

`config_file` must contain the analysis control options as defined in Section 3 of `flintman.pdf`, along with the list of filenames comprising your project. The list of sourcefiles may be obtained as described in Step 1 of Section 7.3 above. Only one item is allowed per line in the config file.

If you want a cross reference, be sure the `.cfg` file has `-Xline`; for call tree, add both `-Xline` and `-t`.

`filtflint` will generate up to four results files ending with `.lnt.filt`, `.xrf.filt`, `.tre.filt`, and `.stt.filt`. The file stem will be as specified by `-S` or `++` in `config_file`, or you can add `-S` and/or `-O` along with a new file stem to the `filtflint` command line.

This is a lot to take in, but for large projects, we’re confident the time you invest here will pay off ten- or one hundred-fold! If you have any questions, just email [support@cleanscape.net](mailto:support@cleanscape.net).

### D. Sample session using `filtflint`

In the sample Linux command line session below, we invoke Flint to output a set of reports, then invoke `filtflint`. Commands we enter are colored green.

Because we use `++` on the Flint command line, results are split into four files according to the functionality of `++`: `flint.lnt`, `flint.xrf`, `flint.tre`, and `flint.stt`. See Section 3 of `flintman.pdf` if you’re interested in what the other command operands are.

Since the input is split, add `-S` to `filtflint`; it will use the stem `flint` as provided with `-L` and the resulting filtered output will be `flint.lnt.filt`, `flint.xrf.filt`, `flint.tre.filt`, and `flint.stt.filt`.

```

CentOS7: ~/test$ flint -+egstXline *.f *.F *.F90

CentOS7: ~/test$ ls -l flint.*

-rw-rw-r--. 1 frank 18836399 Jul 16 14:52 flint.lnt
-rw-rw-r--. 1 frank 4691 Jul 16 14:52 flint.stt
-rw-rw-r--. 1 frank 7588162 Jul 16 14:52 flint.tre
-rw-rw-r--. 1 frank 59618216 Jul 16 14:52 flint.xrf

CentOS7: ~/test$ flint -S -L flint -U vector.f

Ready to run. Configuration:
Use previous analysis in /home/frank/testflint.lnt
 /home/frank/testflint.xrf
 /home/frank/testflint.tre
 /home/frank/testflint.stt
Output file: flint.[lnt.filt,xrf.filt,tre.filt,stt.filt]
Filter for: vector.f

Processing 1,378,450 lines; please wait...
- Filtering analysis results (459,111 lines)
- Filtering global analysis results (456 lines)
- Filtering cross reference (918,767 lines)
- Generating filtered call tree
- Generating filtered summary (includes full stats for overall project health)
DONE. See above for output file name(s).

CentOS7: ~/test$ ls -l flint.*.filt

-rw-rw-r--. 1 frank 28180 Jul 16 14:56 flint.lnt.filt
-rw-rw-r--. 1 frank 2150 Jul 16 14:56 flint.stt.filt
-rw-rw-r--. 1 frank 718 Jul 16 14:56 flint.tre.filt
-rw-rw-r--. 1 frank 163025 Jul 16 14:56 flint.xrf.filt

```

As you can see in the above run, the cross reference output for the entire project was ~60 MB, while the filtered result for the UUT of interest was only 163k – a 99.9973% size reduction!

Here are the detailed results for a real-world project:

|                                                                                                                               |                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <i>Test case description</i>                                                                                                  | Quantum chemistry solver<br>331 files<br>65MB of source<br>1.5MLOC                          |
| <i>Analysis Time</i>                                                                                                          | 36.0 sec on quad-core i7 Windows 7 workstation<br>with 16 GB memory                         |
| <i>Working set (memory) size</i>                                                                                              | 37.6 MB                                                                                     |
| <i>Unfiltered results, initial whole program run. Includes complete analysis, full cross reference, call tree, statistics</i> | 1,494,904 lines<br>– 911,772 lines were the cross reference<br>– 181,297 were the call tree |
| <i>Filtered results for whole program, single UUT sourcefile</i>                                                              | 3611 lines<br>– 2886 lines were the cross reference<br>– 87 were the call tree              |
| <i>Key Benefit</i>                                                                                                            | 0.242% filtered/unfiltered lines →<br>Only 581 analysis lines to review (out of 459,997)    |