COMMAND LINE REFERENCE MANUAL

Cleanscape FortranLint^â

SOURCE CODE ANALYZER

Version 7.59

Cleanscape Software International

COMMAND LINE REFERENCE MANUAL for

Cleanscape FortranLint$^{\hat{a}}$, A SOURCE LEVEL CODE ANALYZER

For FORTRAN PROGRAMMING

On Unix, Linux, Windows, Mac, FreeBSD, and VMS Systems

Note: Licensed users may photocopy for distribution.

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

Copyright 1987-2023

**CLEANSCAPE
NOTICE OF COPYRIGHTS**

�֎✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦✦
✦✦✦

Table of Contents

1

❖❖
❖❖

Introduction

1.1 Overview

FortranLint is a programming tool that simplifies the debugging and maintenance of FORTRAN 77, Fortran 90, Fortran 95, and Fortran 2003/2008 programs. Fortran-lint maps well to compiler capability, non-object-oriented.

FortranLint includes a source code analyzer that can detect a wide range of potential problems, including:

- Inappropriate arguments passed to functions

- Inconsistencies in common block declarations

- Non-portable code

- Type usage conflicts across different subprograms/program units

- Unused functions, subroutines, and variables

- Variables that are referenced but not set

FortranLint can be used to:

- Check source files before they are compiled

- Isolate obscure problems

- Identify problems before debugging is required

- Map out unfamiliar programs

- Enforce programming standards

The diagnostic messages produced by FortranLint are more detailed than those produced by standard compilers, and cover a wider range of problems. FortranLint analyzes source files both individually and as a group, and can therefore identify problems that are beyond the scope of a compiler.

1.2 Latest Functional Improvements in Version 7

New changes in version 7.57/7.58/7.59 are in *italic.*

Language Additions

- BLOCK construct (F08)
- DO CONCURRENT (F08)
- *IMPORT statement (F03)*
- *ENUM, END ENUM and ENUMERATOR statements (F03)*
- *PROCEDURE statement (non-oo) (F03)*
- *ABSTRACT INTERFACE statement (F03)*
- *FINAL statement (F03)*
- VALUE statement and attribute
- VOLATILE statement and attribute
- BIND statement and attribute
- Language binding can be specified in FUNCTION, SUBROUTINE, and ENTRY statements
- *"PROCEDURE" as shorthand for "MODULE PROCEDURE" (F03)*
- Pointer objects can now have the INTENT attribute
- Symbol names up to 63 characters
- Statements up to 256 lines (16,000-character limit for Flint)
- Square brackets [ ] are permitted to delimit array constructors in addition to previous standard / /
- Binary, Octal, and Hex (BOZ) constants to intrinsic functions INT, REAL, DBLE, and CMPLX

Intrinsics

- ISO_C_BINDING and ISO_FORTRAN_ENV
- IEEE_ARITHMETIC, IEEE_EXCEPTIONS, and IEEE_FEATURES
- INTRINSIC and NON-INTRINSIC can be specified for modules in USE statements
- GET_COMMAND, GET_COMMAND_ARGUMENT, COMMAND_ARGUMENT_COUNT, and GET_ENVIRONMENT_VARIABLE intrinsics
- F08 intrinsic functions EXECUTE_COMMAND_LINE, COMPILER_OPTIONS, and COMPILER_VERSION
- *F08 intrinsic functions BGE, BGT, BLE, BLT, DSHIFTL, DSHIFTR, LEADZ, POPCNT, POPPAR, TRAILZ, MASKL, MASKR, SHIFTA, SHIFTL, SHIFTR, MERGE_BITS, IALL, IANY, IPARITY, STORAGE_SIZE, BESSEL_J0/1/N, BESSEL_Y0/1/N, transformational forms of BESSEL_JN/BESSEL_YN, ERF, ERFC, ERFC_SCALED, GAMMA, HYPOT, LOG_GAMMA, NORM2, PARITY, and FINDLOC*
- *Updated the following to the F08 standard: ACOS, ASIN, ATAN, COSH, SINH, TANH, TAN, ACOSH, ASINH, ATANH, ASCII kind, MINLOC, and MAXLOC*
- MOVE_ALLOC (F03)
- Updated FORALL and WHERE constructs to the F03 standard
- Character arrays for MAXLOC, MINLOC, MAXVAL, and MINVAL
- KIND= argument may be applied to ACHAR, IACHAR, ICHAR, LEN, LEN_TRIM, MAXLOC, and MINLOC

I/O

- FLUSH statement
- WAIT statement (with ASYNCHRONOUS/ASYNCH and DONE extensions from IBM)
- ASYNCHRONOUS statement and attribute
- The following I/O specifiers have been added or updated to the F03 standard: ACCESS, ASYNCHRONOUS (along with ASYNCH alias, an IBM extension), BLANK, DECIMAL, DELIM, ENCODING, ID, IOMSG, PAD, PENDING, POS, RECORDTYPE (DEC, HP, Intel extension) ROUND, SIGN, SIZE, and STREAM
- F08 I/O specifier NEWUNIT
- Any KIND is permissible with integer specifiers (SIZE, NEXTREC, etc.)
- Intrinsic functions IS_IOSTAT_END, IS_IOSTAT_EOR
- Comma after a P-edit descriptor is optional when followed by a repeat
- NEW_LINE intrinsic function
- SELECTED_CHAR_KIND intrinsic function
- F08 extension to SELECTED_REAL_KIND intrinsic function

Other Additions/Improvements

- A new statistics option, -A (/ASSESS), provides sorted list of procedure names, line counts, *parameter counts,* and cyclomatic complexities v(G), output in the summary/statistics section
- *A new message, #909, that can be expressed when v(G), line count, or parameter count exceeds user-specified maximum (-AVn, -ALn, -APn)*
- *Magic comments which allow the user to Omit message(s) for the following line of source (-MMagicCom/-MMagicShow)*
- Complete MPI interface definition using stubs for thorough interface analysis and variable tracking (see section 9.2 for stub information)
- Complete NetCDF interface definition (F77 format)
- Force -O+ processing. Example: --w no longer suppresses explicit -O+103 (warning: dead code).
- *Force Flint to ignore sequence numbers and any characters past col. 72 (*`-Mstrict72`*)*
- Flint now exits immediately if source is >72 columns but no -e was specified, which generated a lot of spurious messages
- *Flint now exits immediately if include files are missing, which generated a lot of spurious messages*
- *Two pre-analysis modes which check for common issues that can generate spurious errors, especially when using Flint for the first time (see section 7.4 of flintguide.pdf)*
- Some messages downgraded to FYIs if analysis performed in local mode (i.e., no -g): #136, 538
- New xref legend item, 'C' for "assoCiated"
- A nother new xref legend item, 'c' (lowercase) for symbol used as loop counter
- *Yet more xref legends: lowercase 's' and 'r' for targets that are set or refenced via their pointers; also eliminates misleading #742/126 messages regarding target referenced but never set*
- A new command line switch -o `"message_format"` (use single quotes on *nix) to change the format of Flint's analysis messages, suitable for interfacing to IDE/ editors such as Visual Studio
- Improved pointer handling
- Significant memory management updates
- Native 64-bit builds
- *New analyses: now 911 unique messages and 1662 scenarios where these messages can be applied*
- *Fug bixes*
- *Various operational and performance enhancements*

<u>**User Interface Changes to Note**</u>

- Separator for cross reference (xref) content selection is now '.' instead of '_'; see Section 8.4. Prior versions of Flint retain the old syntax.

- *In building 64-bit versions, the FDB database required storing and retrieving 64-bit pointers, necessitating a restructure of the binary file format. Notify support@cleanscape.net for assistance if this affected you.*

- *As of version 7.57, messages are enabled for intrinsic procedures. In prior versions, if you did not specify* `-g` *you would not get any messages regarding use of the intrinsic (passing a CHARACTER when INTEGER is expected, for instance) because these are considered interface issues, and interface issues are suppressed unless* `-g` *is specified. In v7.57, these messages are enabled by default. To revert to pre-7.57 behavior, add* `-Moldintrinmsgs` *to your Flint command line.*

1.3 Document Scope

This is the command line reference manual that gives full descriptions of all commands available in FortranLint. All users, no matter what user interface mode they use, should reference this document for full descriptions of each command.

Chapters 11-16 also describe the use of Xlint, an adjunct graphical source browser available on Unix and Linux only.

The FortranLint GUI (an executable called flintgui), available for Unix, Linux, Mac, and Windows, is a front-end to the command-line FortranLint product. The command line option used to fulfill a GUI operation is detailed in the GUI's online help reference.

For installation and usage of the Flint GUI, consult the flintguide.pdf file located in the 'doc' subdirectory.

1.4 Documentation Style

Command descriptions for Unix, Linux, Mac, FreeBSD, and Windows are all the same (except where noted) and may be prefaced with the shorthand description, "UNIX". Command descriptions for VMS are explicitly referenced with the text, "under VMS".

Flint with capital 'F' may sometimes be used as a shorthand reference to FortranLint. flint in all small letters is the instruction to use at the command line.

Commands are in Arial (or Helvetica) typeface.
FortranLint output is in `Courier New` typeface.

"Flint" (like the city in Michigan or our favorite movie, *In Like Flint*) is a shorthand notation for FortranLint and may be used in this document.

2

❖❖
❖❖

Getting Started

2.1 Installation

> **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 <u>sales@cleanscape.net</u> within 15 days of purchase to arrange for return.**

For installation instructions, see Appendix A, Appendix B (VMS only), or the separate FortranLint Quick Start Guide (flintguide.pdf in the 'doc' subdirectory).

2.2 Analyzing Programs

To run FortranLint, use a command of the form:
        flint -*options* file1.f file2.f file3.f
or
        flint /*options* file1.for file2.for file3.for   under VMS

where *options* may be one or more options, and each of the specified files is a FORTRAN source file containing any number of FORTRAN program units. Options may be intermixed with or appear after file names.

If FortranLint is invoked without any options or parameters, a "help" screen will be displayed:
        flint

If source files are specified, but no options are given, FortranLint will perform a basic analysis of the source files and output the results to the console.

For example, to analyze a single source file, use a command of the form:
        flint demo.f
or
        flint demo.for   under VMS

The following commands will perform a more detailed analysis:
        flint -fgs demo.f
or
        flint /FYI /GLOBAL /STATISTICS demo.for   under VMS

2.3 Managing the Output

When FortranLint is used on a large program for the first time, it may report hundreds or thousands of inconsistencies. FortranLint has several features that simplify management of the output.

In addition, look to our [website](#) for supplemental information on using Flint. Topics include:
- Specifying the correct sourcefile list for large projects or 3rd party code
- Iterative analysis approach for first time analysis of large projects
- Specify only the necessary subset of files for analyzing one (or a few) sourcefiles
- Memory leak detection
- Using supplied .lsh files for thorough interface analysis when using IEEE modules, MPI, or NetCDF

2.3.1 Severity Levels

If you find the number of messages too unwieldy to start with, first run with just errors being reported. Add `--w` ("minus minus w") to your command line, which disables warnings (warnings are enabled by default in file `$FLINTHOME/flint.cfg`).

Make sure you have not enabled any of the other Diagnostic options as listed on the Flint help screen. Then, after you have dealt with errors-only, remove `--w` from your command line to reenable warnings, deal with those messages, and only then add other diagnostics, one at a time.

2.3.2 Redirection

The command-line option "-S*name*" or "/SPLIT=*name*" (under VMS) will cause FortranLint to redirect output from the console to the following files:

Analysis output	*name*.lnt
Statistics (-s)	*name*.stt
Call tree (-t)	*name*.tre
Cross-reference (-x)	*name*.xrf

Under VMS:

Analysis output	*name*.lnt
Statistics (/STATISTICS)	*name*.stt
Call tree (/TREE)	*name*.tre
Cross-reference (/XREF)	*name*.xrf

For example, the following commands will analyze **demo.f** (or **demo.for**), send analysis output to **demo.lnt**, and send statistics output to **demo.stt**:

        flint -fgs demo.f -Sdemo

or

        flint /FYI /GLOBAL /STAT demo.for /SPLIT=demo   **under VMS**

2.3.3 Statistics Output

The command-line option "**-s**" or ⁄**STATISTICS** (under VMS) enables statistics and related output.

If this option is used, FLINT displays a screen after analysis is completed which includes I⁄O statistics, structural statistics (subroutine counts, etc.) and a list of the error messages, ordered by frequency of occurrence.

2.3.4 Summary Mode

The command-line option "**-+**" or ⁄**SUMMARY** (under VMS) combines three operations:

(a) This option displays a progress meter that tracks the progress of FortranLint in real time.

(b) It redirects FortranLint output (as explained in section 2.3.1). By default, "**-+**" (or ⁄**SUMMARY**) redirects the output to files named **flint.lnt**, **flint.tre**, etc. "**-S**" (or ⁄**SPLIT**) may be used to specify a different base name.

(c) It displays an error-message summary (as described in section 2.3.2).

For example, the following commands will analyze **demo.f** (or **demo.for**), display a progress meter, send analysis output to **flint.lnt**, and display an error-message summary after analysis is completed:

        flint -fg+ demo.f
or
        flint /FYI /GLOBAL /SUMMARY demo.for   under VMS

2.4 Call Trees and Cross Reference Tables

FortranLint will optionally generate a diagram of program structure (i.e., a "call tree") and a symbol-table cross-reference.

For example, the following commands will analyze demo.f (or **demo.for**), output a call tree to the file **demo.tre**, and output a cross-reference to the file **demo.xrf**:

        flint -tx demo.f -Sdemo
or
        flint /TREE /XREF demo.for /SPLIT=demo   under VMS

For additional information on call trees, see chapter 7. For additional information on cross-reference tables, see chapter **8**.

3

❖❖
❖❖

Command Reference

3.1 Command-Line Options

3.1.1 Command Format

To run FortranLint, use the command **flint**, followed by zero or more option switches and one or more file names:

flint [options] [file1 [file2...]] [file3.lbt...] [file4.fdb...]

"**file1 file2**..." are FORTRAN source files. "**.lbt**" files are optional call-interface library files (explained in chapter 9). "**.fdb**" files are optional Xlint database files (explained in chapter 13).

If no options or file names are specified, flint will display a "help" screen.

FORTRAN source files may use any valid FORTRAN filename extension. "**.f**" is a special case; , if a source file has the "**.f**" extension, FortranLint will run the 'C' preprocessor on the file before analyzing it.

Option switches may be specified in any order, and may be intermixed with filename arguments.

3.1.2 Option Format

Options are specified by single-character switches; for example, "-**x**". Lower-case options take no arguments, and may be combined into a single switch. For example, "-**stx**" is equivalent to "-**s** -**t** -**x**". Upper-case options require one or more arguments; these options cannot be combined.

Arguments are specified for UNIX and Windows switches as follows:

-P argument	single- argument switches

or

-P arg1,arg2,arg3, ...	multi- argument switches

Under VMS, options are specified by "word" switches (for example, ∕**XREF**). "Word" switches are not case-sensitive. They may be abbreviated, provided that the abbreviations are unique. For example, ∕**XREF** is an abbreviation for ∕**XREFERENCE**.

Arguments are specified for VMS switches as follows:

/PORT=argument   single‑argument switches

or

/PORT=(arg1,arg2,arg3, ...)   multi‑argument switches

Note: Under VMS, switches should **not** include spaces.

Switch arguments are cumulative. For example, the following commands are equivalent:

```
flint -O 123 -O 200,375      foo.f
flint -O 123,200,375         foo.f
```

Under VMS, these commands are equivalent:

```
flint /SUPPRESS=123 /SUPPRESS=(200,375)      foo.for
flint /SUPPRESS=(123,200,375)                foo.for
```

To disable an option, add an extra dash to the option switch. For example, "-**w**" enables warning messages and "--**w**" disables them.

To disable an option under VMS, add the word "**NO**" to the option switch. For example, /**WARN** enables warning messages and /**NOWARN** disables them.

When an option is disabled, arguments accumulated up to that point are discarded. If the option is re-enabled subsequently, it "starts over". For example, the following commands are equivalent:

```
flint -P ANSI,CRAY  --P  -P SGI    foo.f
flint -P SGI                       foo.f
```

Under VMS, these commands are equivalent:

```
flint /PORT=(ANSI,CRAY) /NOPORT /PORT=SGI    foo.for
flint /PORT=SGI                              foo.for
```

Including quoted strings on the command line must account for quote-handling by the command processor, and differs depending on the host; here are our recommendations:

Unix/Linux: `-DINCFILE="\'lst3.inc\'"`   escaped single quotes in double quotes

Windows: `-DINCFILE="'foo.inc'"`   single quotes enclosed in double quotes

Configuration files may be used to set default values for options. The FortranLint package includes a predefined configuration file named **flint.cfg**; for additional information, see section 3.3.

3.1.3 List of Options

FortranLint options are listed below:

-a, /ANSI

Description: Reports non-ANSI constructs. If FortranLint is run in
FORTRAN 77 mode, this switch has the same effect as
"-**P ansi77**" (or /**PORT=ansi77**).

If FortranLint is run in Fortran 90/95 mode, this switch has the
same effect as "-**P ansi90**" (or /**PORT=ansi90**).

Note 1: To set the language mode, use the -**7**, -**9** and/or /**LANG**
switches.

Note 2: ANSI is limited to F77 or F90. Use -**7** or -**9** explicitly if
you intend to perform ANSI checks.

Syntax: -a
VMS syntax: /ANSI

-A, /ASSESS=

Description: Provides assessment information about the procedures
(subroutines, functions, and program) found in the source code.
This data appears in the statistics report; specifying -A implies -s.

The user must specify how the data is sorted by using suboption S
and supplying a single letter from the table below:

O	original order of Occurrence
N	sort by procedure Name, ascending
T	sort by procedure Type (subroutine, function, program)
L	sort by procedure Line count, descending
P	sort by procedure Parameter count, descending
V	sort by v(G) (cyclomatic complexity), descending

Starting in version 7.58, it is also possible to set a maximum for a
procedure's v(G), parameter count, or line count. If any procedure
exceeds the specified maximum, informational (FYI) message 901
is generated to alert the user. The feature is enabled as follows:

Vn	generate message if v(G) exceeds n
Pn	generate message if Parameter count exceeds n
Ln	generate message if Line count exceeds n

Syntax: -A<S*sort_option*>[, V*number*][, P*number*][, L*number*]
VMS syntax: /ASSESS=S*sort_option*
Example: -ASV,V9,P6,L100

-B, /DATABASE=

Description: Creates a specified database (**.fdb**) file. FortranLint and Xlint use database files to regenerate call trees, cross-reference tables, and diagnostic messages. For additional information, see chapter 13.

Note: FortranLint adds the "**.fdb**" filename extension automatically.

Syntax: -B *file*
VMS syntax: /DATABASE=*file*

-d, /DLINES

Description: Source lines starting with "**D**" in column one (or "**Y**", for EPC code) are "debug" lines.

By default, "debug" lines are treated as comment lines. If "**-d**" (or **/DLINES**) is specified, FortranLint will process "debug" lines along with normal source code.

Note: This option is valid only for fixed-form code.

Syntax: -d
VMS syntax: /DLINES

-D

Description: (UNIX/Windows only.) Defines symbols for the 'C' preprocessor. Applies only if source files are preprocessed ("**.f**" filename extension or "**-p**" option).

For additional information, see section 4.3.

Syntax: -D *symbol*[=*value*], ...
VMS syntax: *N/A*

-e, /EXTEND

Description: By default, if the source format is fixed form, characters past column 72 are ignored. If this option is specified, the source-line width is extended to 132 columns.

For additional information, see section 4.1.

Syntax: -e
VMS syntax: /EXTEND

-E, /FILES=

Description: Reads a specified file and adds its contents to the FortranLint command line.

The file may contain source-file names and/or command-line option switches. Entries may be separated by commands, new lines, or spaces, and may be specified in any order.

Nested expansions are allowed, i.e., the specified file may use the "-**E**" (or **/FILES**) option to process lower-level files.

Wildcards are not supported. I.e., the specified file cannot include entries of the form *.**for**
This option cannot be suppressed, i.e., "--**E**" and **/NOFILES** are not supported.

For additional information, see section 3.3.

Syntax: -E file, ...
VMS syntax: /FILES=(*file*, ...)

-F, /FLOW

Description: Enables local dataflow analysis.

For more information, see Section 5.6.

Syntax: -Fon
VMS syntax: /FLOW=on

/FORM=

See "-**R**".

-f, /FYI

Description: Enables FYI (or "for your information") diagnostics.

FYI diagnostics are informational messages that may (or may not) indicate problems.

Syntax: -f
VMS syntax: /FYI

-g, /GLOBAL

Description: Global analysis. **This option is strongly recommended after initial error messages have been assessed and eliminated.**

By default, subprograms are processed on an individual basis, and call interface checking is not performed. The "**-g**" (or **/GLOBAL**) option enables "global" analysis. If this option is used, FortranLint checks for inconsistencies between subprograms; for example, invalid arguments or common-block problems. This option also improves usage checking and enhances cross-reference output.

Syntax: -g
VMS syntax: /GLOBAL

-i, /INCLUDE

Description: Expands INCLUDE files in source listings. This option applies only when source listings are enabled (see "**-l**" or **/LISTING**).
Syntax: -i
VMS syntax: /INCLUDE

-I, /PATH=

Description: Adds one or more directories to the include-file search list. This switch affects both INCLUDE files and "#include" files.

For additional information, see sections 4.2 and 4.3.

Syntax: -I *path*, ...
VMS syntax: /PATH=([*path*], ...)

Example: -I ../myftn,/usr/sam/headers
/PATH=([FTNCODE],[USR.HEADERS])   *under VMS*

/IMPLICIT

See "**-m**".

/LANG=

See "**-7**" and "**-9**".

-l, /LISTING

Description: Outputs a source listing with line numbers.

Syntax: -l *(lower-case ell)*
VMS syntax: /LISTING

-L,   /LIBRARY=

     Description.   Creates or updates a library template file.

                       This option adds interface information for the current source files to the specified library template (or "**.lbt**") file. "**.lbt**" files may be used to speed up subsequent runs. See chapter 9 for additional information.

                       Note: This option causes FortranLint to run in a special mode, bypassing normal analysis. Consequently, **input files must be free of errors before this option is used**.

     Syntax: -L *file*.lbt
     VMS syntax: /LIBRARY=*file*.lbt

     Example.   flint -L vmslib.lbt vmslib.lsh
                       flint /LIBRARY=mylib.lbt mylib.for   *under VMS*

/LPP=

     See "**-Y**".

-m,   /IMPLICIT

     Description.   Reports the use of implicit data typing.

     Syntax: -m
     VMS syntax: /IMPLICIT

-M,   /MISC=   (F90 only)

     Description.   Miscellaneous options:

allcalls	Output all "Calls" and "Called by" references for each procedure in the cross-reference report.
ansi_maxloc	Modifies the rules used for HPF checking. For additional information, see section 4.9.
cpp:"*opt*"	Pass *opt* through to the preprocessor. Example: -Mcpp:"-U__FOO__"
depend	If this sub-option is specified, Fortran 90 source file order is irrelevant. Note: This sub-option adds an extra pass, which reduces processing speed slightly.
depend:fname	FortranLint will output the sorted file list and the file dependencies via USE association to the specified file. File extension "**.dep**" is added automatically.

	If the source files are in order, **depend** is not required.
help	Outputs a "help" screen describing these sub-options.
hpf	Enables HPF checking. For additional information, see section 4.9.
ignore_log	Ignore VMS logicals inside INCLUDE statements. Useful when VMS Fortran files are processed under Windows or UNIX. To specify the INCLUDE directories which should be used locally instead of the "logicals", add them using -I.
libcom	Check source-level common blocks against common blocks declared inside FDB libraries; see section 10.4.
libext	Do not search FDB files for unresolved procedures – treat such procedures as externals instead. See section 10.4.
magicCom	Process magic comments inside source code. Presently supports only Omit (-O). *Comment format:*

```
!FLINT –O<msgnum>[,msgnum2,…] ! Commentary
```

Comment rules:
- Must immediately precede sourceline generating the Flint message – no intervening blank/comment lines
- May be used inside INCLUDEd files but cannot be processed inside #include'd files

magicShow	Print info message when magic comment suppresses normal Flint output. Implies magicCom.
noexit	For UNIX users who use shell scripts to check FortranLint results. This sub-option tells FortranLint to return zero unless errors were detected.
oldintrinmsgs	Use pre-version-7.57 behavior and do NOT output messages for intrinsics if global mode (-g) has not been specified.
omp	Perform a heuristic analysis of common OpenMP problems. It is especially suited for customers moving from sequential to parallel code. This

feature is licensed separately; contact Cleanscape for additional information.

path_ignore   Ignores directory paths inside INCLUDE statements. Useful when Fortran files are moved from one machine to another. Use –I to specify what directories should be used.

strict72   Force Flint to process the sourcecode as 72 columns and ignore sequence numbers/ any characters past column 72.

uselbt   Modifies the precedence rules used for library template (**.lbt**) files. For additional information, see section 9.3.

Syntax: -M *option, ...*
VMS syntax: /MISC=(*option, ...*)

/NOI4

See "**-2**".

-o

Description: Reformat Flint's output with flexibility as to placement of the sections. Valid entries and their meanings are:

$ = Space
F = Filename (fully qualified)
L = line number
C = Column number
S = Severity (error, warning, info)
T = Type (syntax, interface, usage, portability, I/O, internal)
# = Error number
M = Message content string
Any other characters go into output string at that location.

Example: Integration with Microsoft Visual Studio. According to http://blogs.msdn.com/b/msbuild/archive/2006/11/03/msbuild-visual-studio-aware-error-messages-and-message-formats.aspx, a message in this format:

```
filename(line#,col#) : Error #123 : This is some text
```

directed to its output window will be picked up automatically such that double-clicking the message will cause VS' internal editor to jump to the "offending" sourceline.

To accomplish this in Flint, use the following command parameters:

```
-o "F(L,C)$:$S$T$#$:$M" -W199
```

which output two lines, first source, then analysis message:

```
      ISTAT = PRINTIT( CURITEM, 1)
c:\progra~2\cleanscape\flint\examples\demo.f(49,16)
: ERROR Interface #95 : this name is defined as a
subroutine.
```

Notes:
(1) On Unix/Linux, put the format string in single quotes to prevent command-line parsing, or escape any '$' characters. On Windows, encasing in double-quotes is recommended.
(2) Use `-W199` if your editor/IDE expects message all on one line.
(3) At present, the message text can only be output at the end. If this is an issue, email support@cleanscape.net.

Unix Syntax:	-o 'message_format'
Windows Syntax:	-o "message_format"
VMS syntax:	N/A

-O, /SUPPRESS=

Description: Disables or enables individual diagnostic messages.

Syntax:
-O *msg#,msg#, ...*	disables messages by number
-O +*msg#,+msg#, ...*	enables messages by number
-O all	disables all numbered messages
-O +all	enables all numbered messages
-O *msg#,+msg#, ...*	disable/enable can be mixed

VMS syntax:
/SUPPRESS=(*msg#, ...*)	disables messages by number
/SUPPRESS=(+*msg#, ...*)	enables messages by number
/SUPPRESS=ALL	disables all numbered messages
/SUPPRESS=+ALL	enables all numbered messages
	disable/enable can be mixed
/SUPPRESS=(*msg#,+msg#, ...*)	

/OUTPUT=

Description: (VMS only.) Redirects output to a specified file.

Note: use standard-I/O redirection (**flint ... > foo.out**).

Syntax:	N/A
VMS syntax:	/OUTPUT=*file*

See also: **-S** or **/SPLIT**

-p

Description: (UNIX/Windows only.) Send all source files through the 'C' preprocessor.

Note 1: Using '-*p*' is NOT recommended, as it invokes the preprocessor for **every** file specified! Rather, use the Fortran convention of capitalizing the extension (e.g., .FOR) to invoke the preprocessor on a per-file basis.

Note 2: As of ver. 7.51, FortranLint sends any file whose extension starts with **.F** through the 'C' preprocessor, whether or not the '-*p*' option is selected (prior to 7.51, it was just files with the .F extension).

Note 3: On Windows or any OS with case-insensitive filenames, you can capitalize a file's extension on the command line or in a .cfg file to invoke the 'C' preprocessor as described in Note 2.

Syntax: -p
VMS syntax: N/A
See also: -# *(Unix/Windows only)*

-P, /PORT=

Description: Checks for portability issues related to one or more compilers or FORTRAN dialects. In other words, "I plan to port my code to the specified compiler/standard; what issues will there be?"

Supported environments include:

ANSI77 (FORTRAN 77)	NCUBE
ANSI90 (Fortran 90)	OS32 (Concurrent)
CRAY	SGI
CVF (Compaq Visual Fortran)	SUN
HPUX	TRU64
EPC	VAXULTRIX
LAHEY (Windows/Linux)	VMS

For additional information, see sections 4.5 through 4.7.
NOTE: Modern compilers often offer extensions ranging from F90 to F03 (e.g., Sun's compiler for F95 offered the BIND command), so in general Flint will work best by **not** specifying portability options. If you have a particular issue with your compiler, please email support@cleanscape.net .

Syntax: -P *system, ...*
VMS syntax: /PORT[ABILITY]=(*system, ...*)

-q, /QUIT

Description: This option is related to FortranLint 's license manager (see Appendix C). By default, FortranLint waits for a free license, if none is available. If "-**q**" (or ∕**QUIT**) is specified, FortranLint terminates immediately, in this case.

Syntax: -q
VMS syntax: ∕QUIT

-R, /FORM=

Description: When Fortran 90 sources are processed, FortranLint normally determines the source format (fixed or free) based on the filename extension. "-**R**" (or ∕**FORM**) may be used to specify the source format explicitly. For additional information, see section 4.1.

Note: This option does not apply to FORTRAN 77 code.

Syntax: -R fixed   Specifies fixed form
               -R free   Specifies free form

VMS syntax: ∕FORM=fixed   Specifies fixed form
               ∕FORM=free   Specifies free form

-s, /STATISTICS

Description: Enables statistics and related output. If this option is used, FLINT displays a screen after analysis is completed which includes I∕O statistics, structural statistics (subroutine counts, etc.) and a list of the error messages that occurred most frequently in the source code.

Syntax: -s
VMS syntax: ∕STATISTICS

See also: -+ (or ∕SUMMARY)

-S, /SPLIT=

Description: Sends FortranLint output to a group of text files.

If this option is used, FortranLint stores its output as follows:

            Analysis output   *name*lnt
            Statistics (/STATISTICS)   *name*stt
            Call tree (/TREE)   *name*tre
            Cross-reference (/XREF)   *name*xrf

Where *name* is specified by "-S *name*" or "∕SPLIT=*name*" (VMS).

Syntax: -S *name*
VMS syntax: ∕SPLIT=*name*

See also: -+ (or /SUMMARY)

/SUMMARY

See "-+" at the end of this list.

/SYSTEM=

See "-**V**".

-t, /TREE

Description: Generates a "call tree"; i.e., a structural diagram of the "call" structure used by the source code. For call-tree format options, see "-**T**" or /**TREE**. For additional information on call trees, see chapter 7.

Syntax: -t
VMS syntax: /TREE

See also: -T

-T, /TREE=

Description: Sets call-tree sub-options and generates a call tree. (The "help" sub-option is a special case.)

The following sub-options are supported:

alphabetical   FortranLint normally displays sub-trees using the order in which routines were called. If **alphabetical** is used, sub-trees are displayed in alphabetical order. **alphabetical** may be abbreviated to **alpha**.

To restore the default mode of operation, use "-**T noalpha**" (or /**TREE=noalpha**).

condensed   Merges multiple calls to the same routine. To restore the default mode of operation, use "-**T nocondensed**" (or /**TREE=nocondensed**).

graphics=xxx   Changes the graphics characters used to print the call tree. For additional information, see section 7.3.5.

head:symbol   Generates a call tree starting at the specified symbol.

help	Displays a "help" screen describing the call-tree options. No processing is done, if this sub-option is selected.
nolibrary	Suppresses calls to routines defined in libraries (i.e., "**.lbt**" files). For additional information, see section 7.4.3 and chapter 9.
noundefined	Suppresses calls to undefined routines.
squish	To improve readability, FortranLint normally adds extra white space to call trees. "squish" removes the extra space. To restore the default mode of operation, use "-**T nosquish**" (or **/TREE=nosquish**).
trim	This sub-option merges redundant sub-trees to reduce the size of the output. The configuration file shipped with FortranLint enables **trim**, by default. To disable this option, use "-**T notrim**" (or **/TREE=notrim**). **"trim" is strongly recommended for systems that are low on disk space.** For additional information on call trees, see chapter 7.

Syntax: -T *option*, ...
Note: To set call-tree options without generating a call tree, use
-T *option*, ... followed by "--**t**".

VMS syntax: /TREE=(*option*, ...)
Note: To set call-tree options without generating a call tree, use
/TREE=(*option*, ...) followed by "--**t**".

-u, /USAGE

Description: Enables variable usage checking. For example, this feature detects variables that are referenced, but not set.

The configuration file shipped with FortranLint enables this option, by default. If usage checking is not required for a given project, "--**u**" (or **/NOUSAGE**) may be used to disable this option. Some operations will be slightly faster if usage checking is disabled.

Syntax: -u
VMS syntax: /USAGE

/UNIXHELP

See "-**?**" at the end of this list.

-v, /VERSON

Description: Show version number, full path, MD5 checksum, and build date.

Syntax: -v
VMS syntax: /v

-V, /SYSTEM=

Description: FortranLint normally assumes that the FORTRAN compiler running on the host system will be used. To select a different environment, use this option. In other words, "Assume my code was written for the following host, even though I'm running Flint on a different host".

" -**V**" (or ∕**SYSTEM**) tells FortranLint to assume that a specific compiler (or FORTRAN dialect) will be used.

This allows FortranLint to resolve ambiguous extensions (constructs that look similar, but are handled differently in different environments).

Supported environments include:

ANSI77 (FORTRAN 77)	NCUBE
ANSI90 (Fortran 90)	OS32 (Concurrent)
CRAY	SGI
CVF (Compaq Visual Fortran)	SUN
HPUX	TRU64
EPC	VAXULTRIX
LAHEY (Windows/Linux)	VMS

For additional information, see sections 4.5 through 4.7.

Syntax: -V *system*
VMS syntax: /SYSTEM=*system*

-w, /WARNINGS

Description: Enables "warning" messages.

The configuration file shipped with FortranLint enables this option, by default. To disable warnings, use --**w** (or ∕**NO-WARNINGS**).

Syntax: -w
VMS syntax: /WARNINGS

-W, /WIDTH=

Description: Sets output width in columns. This option affects all output, including diagnostic messages and cross-reference tables.

Any value between 40 and 500 may be used. , the default width is 80 columns. Under VMS, the default width is 80 columns unless **/OUTPUT** is used; in this case, the default width is 132 columns.

Syntax: -W *number*
VMS syntax: /WIDTH=*number*

-x, /XREF

Description: Generates a cross-reference table. For cross-reference format options, see "-**X**" or **/XREF**. For additional information on cross-reference tables, see chapter **8**.

Syntax: -x
VMS syntax: /XREF or /XREFERENCE

-X, /XREF=

Description: Sets cross-reference sub-options and generates a cross-reference table.

The following sub-options are supported:

freeform   Selects a compact variable-width format. This is the default setting.

tabular   Selects a fixed-width (132 column) format.

linenumbers   Locations by line numbers rather than by subprogram.

noequiv   By default, the cross-reference entry for a given variable includes usage information for the associated equivalences, whether or not the variable is used directly. **noequiv** suppresses equivalence usage information.

nolegend   Suppresses the legend that describes line number usage codes.

filters   FortranLint supports cross-reference filters. Filters may be used to generate cross-reference tables for items that meet specific constraints. For additional information, see sections **8.3** and **8.4**.

For additional information on cross-reference tables, see chapter 8.

Syntax: -X *option, ...*
Note: To set cross-reference options without generating a cross-reference, use "-**X** *option, ...*" followed by "--**x**".

VMS syntax: /XREF[ERENCE]=(*option, ...*)
Note: To set cross-reference options without generating a cross-reference, use /XREF=(*option, ...*) followed by "--**x**".

-Y, /LPP=

Description: Sets lines per output page. To disable pagination, use a page length of zero.

The default value is zero for console output and 60 lines per page if
"-**S**", "-+", /**OUTPUT**, /**SPLIT**, and/or /**SUMMARY** are used to redirect output.

Syntax: -Y *number*
VMS syntax: /LPP=*number*

-2, /NOI4

Description: On most systems, integers and logicals are four bytes long, by default.

If "-**2**" (or /**NOI4**) is used, FortranLint interprets INTEGER and LOGICAL as INTEGER*2 and LOGICAL*2. Additionally, integer and logical constants are treated as two-byte values unless they are too large to fit into the smaller size.

Syntax: -2
VMS syntax: /NOI4

-7, /LANG=
-9,
-3

Description: This option may be used to specify the input language (FORTRAN 77, Fortran 90/95, or Fortran 2003/2008). As of version 7, -3 is the default, as most compilers are now very fluid as regards syntax rules between the different Fortran standards.

Syntax: -7   Selects FORTRAN 77
                -9   Selects Fortran 90/95
                -3   Selects Fortran 2003/2008

VMS syntax: /LANG=F77 Selects FORTRAN 77
                /LANG=F90 Selects Fortran 90/95

/LANG=F03 Selects Fortran 2003

Note 1: Specify -**7** or -**9** explicitly if you intend to perform ANSI (-**a**) checks;
ANSI checking is limited to F77 or F90 only.

Note2: Specify -**7** or -**9** explicitly if you intend to perform ANSI (-**a**) checks;
ANSI checking is limited to F77 or F90 only.

-#

Description: Specifies the path to the preprocessor Flint should use. NOTE:
                     Under Unix/Linux, the default is /usr/lib. Since there is no
                     default cpp on Windows, add one to PATH, or use this option to
                     specify.

Syntax: -# *preprocessor_path*
VMS Syntax: N/A
See also: -p *(Unix/Windows only)*

-+, /SUMMARY

Description: The command-line option "-+" or /**SUMMARY** (under VMS)
                     combines three operations:

                     (a) This option displays a progress meter that tracks the progress
                          of FortranLint in real time.

                     (b) It redirects FortranLint output (as explained in section 2.3.1).

                          By default, "-+" (or /**SUMMARY**) redirects the output to
                          files named **flint.lnt**, **flint.tre**, etc. "-**S**" (or /**SPLIT**) may be
                          used to specify a different base name.

                     (c) It displays an error-message summary (as described in section
                          2.3.2).

Syntax: -+
VMS syntax: /SUMMARY

-?, /UNIXHELP

Description: (VMS only.) Displays FortranLint 's "letter" option switches.

                     This option is not supported. To display the "letter" switches,
                     execute **flint** with no parameters.

                     For additional information, see section 3.1.2.

VMS syntax: -? or /UNIXHELP

3.1.4 Using UNIX Switches Under VMS

FortranLint 's "letter" option switches (-*letter*) can be also used under VMS.

"Letter" switches can be used inside **flint** configuration files with no special rules or restrictions. However, if "letter" switches are used on the VMS command line, three rules apply:

      (a) "letter" switches do not include white space
      (b) "letter" switches are limited to one argument per switch
      (c) upper-case switches must be double-quoted

For example, the following VMS **flint** commands are equivalent:

```
flint /PORT=sgi      foo.for

flint "-Psgi"         foo.for
```

To specify multiple arguments for a "letter" switch on the VMS command line, use multiple copies of the switch. For example, the following commands are equivalent:

```
flint /TREE=(condensed,nolibrary)      foo.for

flint "-Tcondensed"  "-Tnolibrary"      foo.for
```

Lower-case "letter" options may be combined into a single switch. For example, the following commands are equivalent:

```
flint /IMPLICIT /XREF /NOI4      foo.for

flint -mx2                       foo.for
```

Additional VMS examples:

```
1)     flint /IMPLICIT    foo.for

       flint -m           foo.for


2)     flint /SPLIT=result /WARNINGS /WIDTH=50      foo.for

       flint "-Sresult" -w "-W50"                   foo.for


3)     flint /ANSI /FYI /GLOBAL /SUPPRESS=(201,202)     foo.for

       flint -afg "-O201" "-O202"                       foo.for
```

3.2 Summary of Options
3.2.1 UNIX/Windows Option Summary

Source configuration options:

-d	Process "debug" lines
-e	Extend source width to 132 columns
-I *path*, ...	Set search path for INCLUDE files
-p	Send source files through preprocessor (CPP)
-R form	Specify Fortran 90/95 source form
-V *system*	Specify FORTRAN dialect
-2	Two-byte integers and logicals
-7	Select FORTRAN 77
-9	Select Fortran 90/95
-3	Select Fortran 2003

Diagnostic options:

-a	Report non-ANSI constructs
-f	Report FYI messages
-g	Enable global processing
-m	Report implicit typing
-O *number*, ...	Suppress individual error messages
-P *system*, ...	Enable portability checking
-u	Check data usage
-w	Enable warnings

Cross-reference options:

-x	Generate cross-reference table
-X *option*, ...	Specify cross-reference sub-options

Call tree options:

-t	Generate "call tree"
-T *option*, ...	Specify "call tree" options

Output format options:

-I	Expand INCLUDE files
-l (lowercase ell)	Generate source listing
-o *"message_format"*	Specify message output format
-W *number*	Set output page width
-Y *number*	Set output page length

Other output control options:

-+	"Progress/summary" mode (implies -**S**)
-A *option*	Provide assessment data in statistics (implies -s)
-B *file*	Create database (**.fdb**) file
-L *file*	Create library (**.lbt**) file
-s	Generate statistics
-S *file*	Split output and redirect it

Miscellaneous options:

-D *definition*, ...	Define preprocessor-level symbols
-E *file*	Expand configuration file
-M *option*, ...	Miscellaneous options
-# *preproc_path*	Preprocessor if different than `cpp` in `$PATH`
-q	Quit if no licenses are available

3.2.2 VMS Option Summary

Source configuration options:

/DLINES	Process "debug" lines
/EXTEND	Extend source width to 132 columns
/FORM=*form*	Specify Fortran 90/95 source form
/LANG=*language*	Specify language (F77 or F90/95)
/NOI4	Two-byte integers and logicals
/SYSTEM=*system*	Specify FORTRAN dialect
/PATH=([*path*], ...)	Set search path for INCLUDE files

Diagnostic options:

/ANSI	Report non-ANSI constructs
/FYI	Report FYI messages
/GLOBAL	Enable global processing
/IMPLICIT	Report implicit typing
/PORT=(*system*, ...)	Enable portability checking
/SUPPRESS=(*number*, ...)	Suppress individual error messages
/USAGE	Check data usage
/WARNINGS	Enable warnings

Cross-reference options:

/XREF	Generate cross-reference table
/XREF=(*option*, ...)	Specify cross-reference sub-options

Call tree options:

/TREE	Generate "call tree"
/TREE=(*option*, ...)	Specify "call tree" sub-options

Output format options:

/INCLUDE	Expand INCLUDE files
/LIST	Generate source listing
/LPP=*number*	Set output page length
/WIDTH=*number*	Set output width

Other output control options:

/ASSESS=*sort_option*	Generate assessment info in statistics (implies -s)
/DATABASE=*file*	Create database (**.fdb**) file
/LIBRARY=*file*	Create library (**.lbt**) file
/OUTPUT=*file*	Redirect output to a specified file
/SPLIT=*file*	Split output and redirect it
/STATISTICS	Generate statistics
/SUMMARY	"Progress/summary" mode (implies **/SPLIT**)

Miscellaneous options:

/FILES=*file*	Expand configuration file
/MISC=(*option*, ...)	Miscellaneous options
/QUIT	Quit if no licenses are free
/UNIXHELP or -?	Display UNIX "letter" options

3.3 Configuration Files

Command-line arguments may be specified indirectly, inside text files. These are called *configuration files*. If **bar.txt** is a text file containing option switches or filenames, the following commands will add the contents of **bar.txt** to the FortranLint argument list:

        flint -E bar.txt   foo.f
or
        flint /FILE=bar.txt   foo.for   under VMS

bar.txt may specify any number of switches or filenames. There are two restrictions:

(a) Arguments must be separated by white space or new lines

(b) Wildcards (such as *.**for**) are not supported

FortranLint may be used for multiple purposes: quick syntax checks, mapping out unfamiliar programs, etc. Configuration files are a convenient way to select different sets of options.

To set FortranLint options automatically, create a configuration file named **flint.cfg** and add option switches to this file.

FortranLint searches for **flint.cfg** in the following directories:

(a) Current working directory

(b) Directories specified by the environment variable **FLINTCFG** or logical **FLINTCFG** (under VMS)

(c) FortranLint installation directory, as specified by the environment variable **FLINTHOME** or logical **FLINTHOME** (under VMS)

Note: Command-line options override options set in **flint.cfg**.

Multiple configuration files may be used; e.g., for different projects. **FLINTCFG** should be set appropriately for users working on each project.

For additional information on **FLINTCFG** and **FLINTHOME**, see section 3.4.

FortranLint does not impose a fixed limit on configuration-file line length. However, system constraints may impose a limit for some environments.

The inclusion of strings in configuration files differs depending on host:

Unix/Linux:	Use escaped single quotes	`-DINCFILE=\'foo.inc\'`
Windows:	Use just single quotes	`-DINCFILE='foo.inc'`

This is the default **flint.cfg** file as of version 6:

```
! Default Flint configuration file. Included by default in every Flint run.
! Note carefully the suppressed messages; -O276 suppresses numeric conversion
! like integer->real (integer->char is detected as error 161).

-w                          ! enable warnings
-u                          ! enable usage checking
-O207                       ! suppress hollerith constant warning
-O276                       ! suppress data type conversion FYI
-O76                        ! suppress mixed mode arithmetic FYI
-O261                       ! suppress initializer data type converted FYI
-Ttrim                      ! make TRIM the default tree format
--t                         ! default is to NOT output tree
-Xno.unreferenced.parameters  ! Eliminate unreferenced parameters
-Xno.unused.common.variables ! Show common variables only where they are used
-Xno.named.IEEE_*           ! Exclude symbols for IEEE intrinsic modules
-Xno.named.C_*              ! Exclude symbols for ISO intrinsic modules
-Xno.named.ISO_*            ! Ditto
-Xno.named.COMPILER_*       ! Exclude F08
COMPILER_VERSION/COMPILER_OPTIONS
--x                         ! default is to NOT output xref
```

Note: VMS configuration files may use "letter" switches without special rules or restrictions. However, several restrictions apply if "letter" switches are used on the VMS command line. For additional information, see section 3.1.2.

For the current set of default options, see the copy of **flint.cfg** provided with Flint.

Usage Hint: If you are expecting, but not seeing, certain messages from Flint, check the **flint.cfg** file in use. One classic example is the automatic conversion of integer to real, as in `r = selected_char_kind('ascii')`. This results in Message #276, but as you can see from above, this message is suppressed by default. (#276 is an FYI only because it is numeric conversion; an integer-to-character conversion attempt warrants Error #161.)

3.4 Environment Variables / Logicals

FortranLint recognizes the following environment variables or logicals (under VMS):

Variable	Description
FLINTCFG	Directory that contains alternate support files (see below)
FLINTHOME	FortranLint installation directory
FLINTHOST	Hostname of system running license-manager daemon
SYS$SCRATCH on	(VMS only) Directory used for temporary files. Flint uses /tmp
	Unix, and %TEMP%, %TMP%, or c:\ in that order on

Windows.

FLINTHOME specifies the location of the main FortranLint directory (i.e., the directory where the flint binary exists). This variable is set during installation (see Appendix A or Appendix B).

Flint includes a license-manager daemon (see Appendix C). **FLINTHOST** specifies
the system where the daemon resides. This variable is also set during installation.

FortranLint uses the following run-time support files:

flint.cfg	Configuration file (see section 3.4)
flint.err	Error messages
flint.hls	"Help" file
ieeea.lsh	IEEE Arithmetic intrinsic module
ieeee.lsh	IEEE Exceptions intrinsic module
ieeef.lsh	IEEE Features intrinsic module
isobind.lsh	ISO C Binding intrinsic module
isoenv.lsh	ISO Fortran Environment intrinsic module
unixlib.lbt	UNIX library definitions (see chapter 9)
vmslib.lbt	VMS library definitions

By default, FortranLint uses the copies stored in the main FortranLint directory (i.e.,
the **FLINTHOME** directory). However, if **FLINTCFG** is defined, FortranLint
searches the **FLINTCFG** directory for support files before it loads the default copies.
Users may set this variable to load customized versions of the support files.

FLINTCFG specifies one or more directories using the following format:
        directory-path
or
        directory-path*<SEP>*…*<SEP>*directory-path
where *<SEP>* is : for Unix, ; for Windows, or   , for VMS.

Users may define **TMPDIR** or **SYS$SCRATCH** (under VMS) to set or change the
directory where FortranLint stores its temporary files.

Note: **TMPDIR** is ignored on UNIX systems that don't support the standard library
routine **tempnam( )**.

4

❖❖
❖❖

Source Conventions

4.1 Source Format

FortranLint accepts one or more FORTRAN source files as input. Each source file
may contain one or more FORTRAN subprograms (or program units). A
subprogram/program unit may be a subroutine, a function, a block data module, or a
main program. INCLUDE-file names should *not* be specified explicitly on the
command line or in configuration files.

FortranLint understands several different source formats. In FORTRAN 77 mode (-**7**
or **/LANG=F77** option), FortranLint assumes ANSI-standard fixed format, with a
continuation indicator at column **6** and a comment field starting at column 73. To
process FORTRAN 77 code that extends past column 72, add the option "-**e**" or
/EXTEND (under VMS).

In Fortran 90/95/03 mode (-**3**/-**9** or **/LANG=F03/F90** options), sources may use
either free format or FORTRAN 77 fixed format. Variable-position comments
(starting with '!') may be used in either fixed or free format. FORTRAN 77-style
comments (starting with a 'C' in column 1) may be used only in fixed format. Free-
format lines may contain up to 132 characters.

TAB formatting is supported for target environments that allow it; i.e., if the label field
contains a TAB character, processing skips to the first non-blank character. If that
character is a non-zero digit, the source line is treated as a continuation line; otherwise,
the line is treated as a statement. TAB-formatted lines may be intermixed with normal
fixed-format lines.

FortranLint normally distinguishes between free-format files and fixed-format files
based on filename extension. By default, "**.f90**" files are assumed to be free format and
other files are assumed to be fixed or TAB format. To override the default setting, use
the "-**R**" option or the **/FORM** option (under VMS). For additional information, see
chapter 3.

The maximum number of continuation lines supported is 1,000 lines per statement, and
there is a maximum of 32,000 significant characters per statement.

4.1.1 "Debug" Lines

Source lines starting with "**D**" in column one (or "**Y**", for EPC code) are "debug" lines. By default, "debug" lines are treated as comment lines. If "-**d**" (or **/DLINES**) is specified, FortranLint will process "debug" lines along with normal source code.

4.2 Include Files

Standard INCLUDE statements are supported. FortranLint searches the following directories for INCLUDE files:

    (a) The directory which contains the source file that the current INCLUDE statement belongs to.

    (b) The user's current directory (at the time when FortranLint was started).

    (c) (VMS only.) The absolute path specified by the INCLUDE statement (taking logicals into account).

    (d) Directories specified by "-**I**" (or **/INCLUDE**) option switches, moving from left to right.

    (e) The standard directory "/usr/include". (UNIX) **–OR–**
the directories in environment variable %INCLUDE% (Windows)

If an INCLUDE file can't be located, FortranLint prints an error message and attempts to continue.

INCLUDE files may be nested up to 10 levels deep.

Note: FORTRAN programs may use both INCLUDE statements and "#include" statements. "#include" is similar to INCLUDE; however, "#include" statements are handled by the 'C' preprocessor. For additional information, see the next section.

4.3 'C' preprocessor (UNIX/Windows only)

FortranLint supports the 'C' preprocessor; i.e., source files may use standard 'C' "#define", "#ifdef", and "#include" statements.

Source files with extensions starting with "**.F**" (.F, .FOR, .F90, etc.) are sent through the preprocessor automatically. If the command-line option "-**p**" is used, FortranLint sends *all* source files through the preprocessor, regardless of filename extension. Preprocessor output is then checked at the FORTRAN level. Line numbers used for error messages are translated appropriately.

By default, FortranLint assumes that the preprocessor is /**usr**/**lib**/**cpp**. To use a different preprocessor, run **flpatch** and patch the **cpp** parameter in the **flint** executable (see Appendix A), or use the -# option.

For Windows users, the first cpp in %PATH% is used. One can be specified in the GUI (see GUI online help) or using the -# command line option.

The option switch "-**D**" may be used to define symbols at the preprocessor level, and the option switch "-**I**" may be used to specify "#include" directories. For additional information, see chapter 3.

FortranLint passes the following command-line arguments to the preprocessor:

    (a) "-**D**" and/or "-**I**" option switches, if any

    (b) FORTRAN source-file name

    (c) Output-file name

Note: Files loaded by INCLUDE statements are loaded directly by FortranLint; i.e., these files are not preprocessed.

4.4 CDD and DBMS Processing (VMS Only)

4.4.1 CDD (Common Data Dictionary) Declarations

FortranLint supports standard DICTIONARY statements.

DICTIONARY is similar to INCLUDE in that it adds declarations to the current routine. However, DICTIONARY differs from INCLUDE in that it takes data structures from a CDD dictionary instead of a source file. FortranLint uses the FORTRAN compiler as a preprocessor to expand DICTIONARY statements into normal code.

4.4.2 DBMS Support (FDML Statements)

FortranLint supports FDML statements (for example, *invoke, ready, use, commit, rollback, disconnect, connect, erase, get, modify, fetch, find, free, also, null, within, keep, reconnect,* and *store*). *invoke* statements are preprocessed by the FORTRAN compiler in the same manner as DICTIONARY statements. FortranLint processes all other FDML statements directly.

Note: Usage checking is suppressed for variables that are created by *invoke* statements.

4.4.3 CDD/DBMS Requirements

FortranLint uses the FORTRAN compiler to expand DICTIONARY and *invoke* statements into normal code. The FORTRAN compiler must therefore be installed before these statements can be processed.

Additionally, the VMS CDD package must be installed before DICTIONARY statements can be processed, and the VMS DBMS package must be installed before *invoke* statements can be processed.

4.5 FORTRAN 77 Extensions

FortranLint 's FORTRAN 77 support is based on the 1978 ANSI FORTRAN 77 standard. FortranLint also supports extensions implemented by the following compilers.

System	Compiler	Dialect code
(ANSI standard)	FORTRAN, ANSI X3.9-1978	ANSI77
Cray YMP UNICOS	CFT77 5.0	CRAY
Alpha/Digital UNIX (OSF1)	DEC FORTRAN 6.0	DECUNIX
VAX/VMS	DEC FORTRAN Version 6.0	DECVMS
HP9000 Series HPUX	FORTRAN/9000 8.05	HPUX
Windows and Linux systems	Lahey Fortran F77	LAHEY
NCUBE	NCUBE Fortran	NCUBE
OS32	Concurrent Fortran	OS32
Silicon Graphics IRIX-4D	3.3 FORTRAN 77	SGI
SunOS / Solaris	Sun FORTRAN 1.4	SUN
VAX Ultrix	VAX FORTRAN	VAXULTRIX

Extensions supported by FortranLint include, but are not limited to, the following:

- Data-type size specifiers (for example, INTEGER*4)
- Records, structures, and unions
- Cray-style and Apollo-style pointers
- Debugging lines with "**D**" or "**Y**" in the first column
- TAB formatting
- In-line comments (both "**!**" and "**;**" styles)
- Long symbol names with non-alphanumeric characters
- Numerous binary, octal, and hex constant formats
- Hollerith constants
- Namelist I/O
- Dozens of system-specific I/O statement specifiers
- Hundreds of intrinsic functions
- All I/O format strings, including embedded expressions
- Abbreviated and symbolic expression operators
- Recursion
- Array sections and array expressions

4.6 Fortran 90/95 Extensions

FortranLint's Fortran 90/95 support is based on the 1992 ANSI Fortran-Extended (Fortran 90) standard. FortranLint also supports extensions implemented by the following compilers. **NOTE:** Modern compilers often offer extensions ranging from F90 to F03 (e.g., Sun's compiler for F95 offered the BIND command), so in general Flint will work best by **not** specifying portability options. If you have a particular issue with your compiler, please email support@cleanscape.net .

System	Compiler	Dialect code
(ANSI standard)	Fortran, ANSI X3.198-1992	ANSI90
Cray Y-MP UNICOS 7.0+	CF90 Release 1.0	CRAY
Compaq Visual Fortran	Intel/Compaq Fortran F90	CVF/INTEL
VAX/Alpha OpenVMS	HP/DEC Fortran 90	VMS
EPC	EPC Fortran 90	EPC
HP9000 Series HPUX	FORTRAN/9000 8.05	HPUX
Windows and Linux systems	Lahey Fortran F90 or F95	LAHEY
Silicon Graphics IRIX 6.1	MIPSpro Fortran 90	SGI
SunOS / Solaris	Sun FORTRAN 1.4	SUN
Alpha/Digital UNIX (OSF1)	DEC Fortran 90	TRU64

In particular, FortranLint supports High Performance Fortran (HPF). For additional information on HPF, see section 4.9.

Note: If FortranLint is used in Fortran 90/95 mode, the FORTRAN 77 extensions are supported, with the exception that debugging lines are not allowed in free format.

4.7 Specifying FORTRAN Dialect

FortranLint normally assumes that the FORTRAN compiler running on the host system will be used.

To select a different compiler, use the "-**V**" option or /**SYSTEM** (under VMS) and specify a dialect code from section 4.5 or 4.6. (For option syntax, see chapter 3.)

To flag code that is not supported by a specific dialect, use "-**P**" or /**PORT** (under VMS), instead.

4.8 Default Sizes

On most systems, integers and logicals are four bytes long, by default. To change the default size, use the option "-**2**" or /**NOI4** (under VMS).

If either of these options are selected, FortranLint interprets INTEGER and LOGICAL as INTEGER*2 and LOGICAL*2. Additionally, integer and logical constants are treated as two-byte values, unless they are too large into fit into the smaller size.

4.9 High Performance Fortran (HPF)

FortranLint supports High Performance Fortran (HPF).

By default, HPF statements are treated as normal comments. To enable HPF checking, use the option "-**Mhpf**" or */**MISC**=**hpf** (under VMS).

To add HPF processors and templates to a cross-reference, enable HPF checking and select **linenumbers** or **tabular** output format:

, use:	-Mhpf -Xlinenumbers
or	-Mhpf -Xtabular

Under VMS, use:	/MISC=hpf /XREF=linenumbers
or	/MISC=hpf /XREF=tabular

For additional information on the **linenumbers** and **tabular** formats, see section 8.3.

For non-DEC target systems, FortranLint normally checks argument lists for MAXLOC() and MINLOC() using the following rules:

        MAXLOC(ARRAY, DIM, MASK)
        MINLOC (ARRAY, DIM, MASK)

ARRAY	must be an integer or real array
DIM	is optional; if present, must be integer scalar
MASK	is optional; if present, must be of local type and conformable with ARRAY

To apply the ANSI X3.198-1992 rules for MAXLOC() and MINLOC(), use the option "-**Mansi_maxloc**" or */**MISC**=**ansi_maxloc** (under VMS). This option disallows the DIM argument.

Note that **ansi_maxloc** does not apply to DEC targets (i.e., Digital Fortran 90).

As of Flint version 7, MINLOC and MAXLOC conform to the Fortran 2003 specification; hpf and ansi_maxloc are left in place as described above for legacy users.

5

Controlling Analysis

5.1 Setting the Scope

To enable global (inter-module) checking, use the "**-g**" option or **/GLOBAL (**under VMS). Global checking analyzes FORTRAN sources as a group; this enables interface checking and improves usage checking of variables passed as actual arguments.

If "**-g**" (or **/GLOBAL**) is not specified, subprograms are processed on an individual basis, and call interface checking is not performed. It is usually best to analyze a new body of code without –g and use it once the local errors are identified/resolved.

5.2 Message Classification

FortranLint checks for the following five general classes of problems:

- Syntax problems

- Subprogram interface problems

- Variable usage problems

- Portability problems

- Implicitly typed variables

Syntax problems are constructs that will not compile or that may be interpreted by the compiler in a different way than the programmer intended. This includes symbol names that have embedded blanks, re-declared or re-dimensioned variables, and poorly structured branches using **GOTO**s.

Interface problems are problems with the interaction between subprograms. This includes inconsistent argument lists in function or subroutine calls, inconsistent common block organization, and unused or missing subroutines and functions.

Usage problems cover improper use of variables and arrays. Variables should be both set and referenced; any deviation from this is flagged. Attempted redefinition of constants in subprogram calls is also flagged.

Portability problems are constructs that are allowed on the host system but are not recognized or are interpreted differently on other systems. This includes structures, pointers, data type length specifiers, and other extensions.

Implicitly-typed variables can be flagged whether or not the "**IMPLICIT NONE**" statement is used. If "**IMPLICIT NONE**" is used, they will be categorized as syntax errors.

FortranLint breaks syntax problems, interface problems, data usage problems, and portability problems down into three levels of severity:

- Error messages are the most serious and indicate that the code will not compile or, probably, will not operate correctly.

- Warning messages flag constructs that may not operate as intended, that may cause intermittent problems, or that may make no sense.

- FYI (or "for your information") messages are used to flag minor issues that may or may not be problems.

5.3 Selecting Analysis Level

Categories of messages may be enabled or disabled using the following options:

- Syntax   Always enabled
- Interface   "-**g**" or **/GLOBAL** (under VMS)
- Usage   "-**u**" or **/USAGE** (This option is on, by default)
- Portability   "-**a**" or **/ANSI**

  -**P***system* or **/PORTABILITY**=*system* (see section 5.5)

- Implicit typing "-**m**" or **/IMPLICIT**

Note: If global interface checking (-**g** or **/GLOBAL**) is enabled, usage checking will detect a wider range of problems.

Severity level of messages in the above categories is controlled with the following options:

- Errors   Always enabled
- Warnings   "-**w**" or **/WARNINGS** (This option is on, by default)
- FYIs   "-**f**" or **/FYI**

To disable a category or level, add an extra dash (e.g., "--**w**") or "**NO**" (e.g., **/NOWARNINGS**) under VMS.

Examples:

To perform a comprehensive analysis, use the options "-**gamf**" or "**/GLOBAL /ANSI /IMPLICIT /FYI**" (under VMS). As of Flint version 6, we recommend not using –a or /ANSI, as most compilers have extensions or allow mixed F77 -> F03 features.

To perform basic syntax checking, use "--**uw**" or "**/NOUSAGE /NOWARNINGS**" (under VMS).

5.4 Suppressing Individual Messages

To suppress individual diagnostic messages, use the "-**O**" (omit) option or **/SUPPRESS** (under VMS).

"-**O**" and **/SUPPRESS** accept message numbers as arguments. Message numbers are shown between the category/severity field and the message text. Multiple instances of the same message have the same number. For additional information, see Appendix E.

"-**O**" and **/SUPPRESS** also accept the word "**all**" as an argument (e.g., "-**Oall**" or **/SUPPRESS=all**). "**all**" suppresses all numbered messages, including syntax errors.

If message numbers (or the word "**all**") are preceded with a plus sign ("+"), the specified message or messages are "unsuppressed". E.g., if "-**O201**" is used to suppress message #201, "-**O+201**" will re-enable it. Note that an unsuppressed message will be shown only if its analysis category and level were selected.

Summary:

-**O** *arg* /**SUPPRESS=***arg*	Action
n	Suppress message #*n*
all	Suppress all messages
+*n*	Unsuppress message #*n*
+all	Unsuppress all messages

Example:

"-**Oall,+279,+281**" or "**/SUPPRESS=(all,+279,+281)**" (under VMS) will suppress all messages but #279 and #281. Since messages #279 and #281 are interface FYIs, the options "-**gf**" or "**/GLOBAL /FYI**" must also be selected in order for these messages to be produced.

5.5 Portability Checking

To check for portability problems (problems that may occur when FORTRAN code is ported to different systems), use the "-**P**" option or /**PORT** (under VMS).

"-**P**" and /**PORT** take target-system names as arguments. Target systems are discussed in section 4.5 (FORTRAN 77 extensions) and section 4.6 (Fortran 90 extensions). System names include ANSI, ANSI90, CRAY, DECUNIX, DECVMS, EPC, HPUX, NCUBE, OS32, SGI, SUN, and VAXULTRIX. Multiple targets may be specified.

To flag non-ANSI constructs, use "-**a**" or /**ANSI** (under VMS). If FortranLint is run in Fortran 90/95 mode, these options have the same effect as "-**Pansi90**" and /**PORT=ANSI90**. Otherwise, they have the same effect as "-**Pansi**" and /**PORT=ANSI**.

Example:

If FORTRAN code is being ported to both VAX/VMS and CRAY systems, use "-**Pdecvms -Pcray**" or "/**PORT=(DECVMS,CRAY)**" (under VMS) to check for portability problems related to either target system.

Note that Flint only performs ANSI checks against the F77 or F90 standards; this is due to creeping featurism in compilers that allow F03 constructs into versions supporting F95 or even F90.

5.6 Local Data Flow Analysis

The ability to track local variables in Fortran code and intelligently report any anomalies in their usage is done with local data flow analysis. Local data flow analysis is superior to sequential inspection of the source code for identifying problems with
> • initialization
> • improper sequencing of set/reference instances
> • identifying dead or wasteful code.

If "-**Fon**" (under UNIX) or "/**FLOW=on**" (under VMS) is added to the command line, the source code will be analyzed to glean more information by analyzing down each of the program's control paths. For example, with this option on, Fortran-lint will inspect both branches of an IF-THEN-ELSE conditional to determine if a variable has been initialized for both branches, and whether a variable has been set before referenced 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.

Additional capability is added from time-to-time; for a complete list of dataflow analysis options, try the -Fhelp option on the command line.

6

❖❖
❖❖

Analysis Output

6.1 Overview

By default, FortranLint sends all text output to the console (**stdout** or
SYS$OUTPUT under VMS). The output is divided into sections, which are
printed in the following order:

Section	Controlled by
Current options	N/A
List of source files	N/A
Source listing	-l -i (/LISTING /INCLUDE)
Analysis output	-g -u -m -P -a -w -f -O (/GLOBAL /USAGE /IMPLICIT /PORT/ANSI /WARNINGS /FYI /SUPPRESS)
Call tree	-t -T (/TREE /TREE=)
Cross-reference tables	-x -X (/XREF /XREF=)
Statistics	-s (/STATISTICS)

To redirect output, use the standard UNIX redirection operators or FortranLint's
"**-S**" and "**-+**" options. To redirect output under VMS, use the options
∕OUTPUT, **∕SPLIT**, or **∕SUMMARY**.

For additional information on "**-S**" and **∕SPLIT**, see section 2.3.1 or chapter 3.

For additional information on "**-+**" and **∕SUMMARY**, see section 2.3.3 or chapter 3.

To modify the output page width or page length, use "**-W**" and "**-Y**" or **∕WIDTH**
and **∕LPP** (under VMS).

6.2 Summary Mode

FortranLint provides an optional progress meter. The progress meter is a
stationary counter (displayed on the console) that tracks the progress of analysis
from 0% to 100%.

To display the progress meter, use "-+" or /**SUMMARY** (under VMS).

By default, these options divert normal **flint** output to a set of text files.
Specifically, enabling the progress meter also sets the option "-**Sflint**" or
/**SPLIT**= **flint** (under VMS). These options send analysis output to **flint.lnt**,
statistics output to **flint.stt**, etc. To specify a different base name, add an explicit
"-**S**" (or /**SPLIT**) option to the command line.

Note: After analysis is complete, FortranLint erases the progress meter and
displays a summary of the messages produced.

For additional information, see sections 2.3.1 and 2.3.3.

6.3 Output Details

6.3.1 Options and Filenames

The first line of the analysis output shows the FortranLint revision number and
the current date and time. The next few lines show the selected options, along
with where they were specified.

- Default options are options that were specified in the **flint.cfg**
  configuration file in the installation directory. These are the system
  defaults.

- User options are options that were specified in a **flint.cfg** configuration
  file in the directory named in the environment variable **FLINTCFG**.
  These are a user's custom defaults.

- Local options are options that were specified in a **flint.cfg** configuration
  file in the local directory. These are usually the defaults for a specific
  project.

- Expanded options are options that were specified in a configuration file
  expanded onto the command line with the "-**E**" or /**FILES** option.

- Command options are options that were placed on the command line.

The selected source file names are shown next, grouped by directory. A source
listing follows (if requested), along with diagnostic messages.

6.3.2 Source Listing

To produce a source listing, use the "**-l**" (dash ell) option or **/LISTING** (under VMS).

By default, the listing does not expand include files. To expand include files, use the "**-i**" option or **/INCLUDE** (under VMS).

6.3.3 Diagnostic Messages

FortranLint generates a diagnostic message for each problem detected within a subprogram/program unit. Each message includes the source line and a pointer to the column where the problem appears. Also shown are the name of the source file, the subprogram/program unit name, the line number, the message category and severity, the message number, and the message text.

A typical message looks like this:

```
>          CALL DIPSTAT (4, CURITEM)
>                               ^
demo.f:PRINTIT
line 43: INTERFACE ERROR #59- constant is changed by subprogram.
```

Messages are generally printed in the order they appear in the source file, and are grouped by subprogram/program unit. Each message group starts with a header consisting of a row of "*" characters followed by subprogram/program unit information. The header looks like this:

```
*************************************************************
Subroutine PRINTIT          File demo.f                Line 39
```

Additional diagnostic messages may be printed after a subprogram/program unit is completely processed or after all subprograms are processed. For example:

```
IMPLICIT- symbols were implicitly typed:  A, AQDATA, DELTI

USAGE ERROR- local variables referenced but never set: J, K

SYNTAX FYI- unused labels: 150
```

6.4 Statistics Output

To generate statistical reports, use the "**-s**" option or **/STATISTICS** (under VMS). Statistical reports include program size, comment density, and diagnostic messages summarized by number, category, and severity.

Program size statistics appear first. The number of source files is shown, followed by the number of lines and bytes of code for the source files, the include files, and the total of the two:

```
Number of source files:  1

Source files:    52 lines,            1314 bytes  (5% comments, 95% code)
Include files:   44 lines,            1052 bytes  (14% comments, 86%
code)
Total parsed:    96 lines,            2366 bytes  (9% comments, 91% code)
```

Counts on "Include files" reflect all appearances of the include files and will be
much higher than that of the include files alone. "Total parsed" is calculated after
all include files are expanded.

Byte counts do not include newline characters.

Comment percentage is based on byte counts and takes both comment lines and
inline comments into account. The comment percentage for include files and total
parsed is calculated after all include files are expanded. This multiplies the weight
of an include file comment by how many times it is included.

A breakdown of subprograms/program units follows:

```
Total subprograms:  7
    Subroutines:    6
    Functions:      0
    Program:        1
    Block Data:     0
    Modules:        0
```

Shown next is a breakdown of the messages produced. Messages are sorted by
frequency of appearance. Displayed for each message are its category, severity,
number, frequency, and message text. Context-dependent fields in the message
text are shown as asterisks ("*").

```
Individual message summary
----------------------------------------------------------------
INTRFC  ERR #57-      2x: too many arguments.
INTRFC  WARN #63-     2x: expression is changed by subprogram.
SYNTAX  WARN #47-     1x: branch into do loop via label *.
INTRFC  ERR #56-      1x: not enough arguments.
INTRFC  ERR #59-      1x: constant is changed by subprogram.
INTRFC  ERR #95-      1x: this name is defined as a subroutine.
```

The number of messages is displayed last, shown both in total and by category and
severity. The code <supp>, meaning "suppressed", is shown for message
categories and severities that were not selected.

```
Total messages:            18


                Errors      Warnings    FYIs
                ---------   ------------   -------
Syntax:            0            1           0
Interface:         8            4           0
Data usage:        2            1           2

Implicit typing:            <supp>
```

If you add Assessment info by adding `-ASV` to the command line, the Statistics report would provide cyclomatic complexity v(G), starting line number, size, parameter count, and procedure type for each procedure in the supplied sourcecode. For sample file demo90.f90, the output of the Assessment would be:

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

The character after -AS specifies the sort order; in this case it was the complexity v(G). Specifying Vm, Pn, or Lo (where n is an integer of suitable value) will also generate message #901 when v(G), Parameter count, and/or Line Count exceeds the specified maximum (m, n, or o, respectively). Message #901 is automatically enabled when one of these suboptions is specified.

6.5 Exit Status

FortranLint return status output is as follows:

On VMS systems:

```
0x18000001:   No errors/warnings/FYIs;
0x18000003:   FYIs produced;
0x18000000:   Warnings (and FYIs) produced;
0x18000002:   Errors (and warning/FYIs) produced;
0x18000004:   Fatal errors caused FortranLint to
              terminate before completion.
```

On UNIX, Linux, and Windows systems:

```
0:  No errors/warnings/FYIs;
1:  FYIs produced;
2:  Warnings (and FYIs) produced;
3:  Errors (and warning/FYIs) produced;
4:  Fatal errors caused FortranLint to terminate before
    completion.
```

Note that that return status 0, 1, or 2 indicates that FortranLint did not detect any errors with the specified options. If "-**Mnoexit**" is used, FortranLint will return 0 (only), unless errors are detected.

7

❖❖
❖❖

Call Trees

7.1 Overview

"Call trees" are diagrams which outline the calling structure used by the FORTRAN input source files. To generate call trees, use the "-**t**" option or ∕**TREE** (under VMS).

A typical call tree (using the default format) looks like this:

```
FORTRAN-lint  (call tree)
This is a primary tree starting at the program 'PROCDAT'

PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT--PRINTIT-+-DIPSTAT--*PRINT*
        |                         |
        |                         +-GETUNIT
        |
        +-PRINT--PRINTIT-+-DIPSTAT--*PRINT*
                         |
                         +-GETUNIT
```

7.2 Tree Options

To modify the call tree format, use the "-**T**" option or ∕**TREE** (under VMS). These switches take one or more sub-options as arguments, specified as follows:

> -T*option1*,*option2*,*option2*, ...

or
> /TREE=(*option1*,*option2*,*option2*, ...)   under VMS

For a list of sub-options, see the next section.

7.2.1 Arguments

"-**T**" and ∕**TREE** accept the following sub-options:

{no}alphabetic   Calls are normally listed using the order in which they occur. This sub-option sorts call trees alphabetically. The **condensed** option is recommended, in this mode.

{no}condensed   Condenses multiple calls to the same routine. If a routine calls the same routine many times, these calls are merged into one call. (For older versions of FortranLint, this is the default mode.)

disable   Disables call-tree output. This sub-option has the same effect as "--**t**" or ∕**NOTREE** (under VMS).

enable   Enables call-tree output. This sub-option has the same effect as "-**t**" or ∕**TREE** (under VMS).

graphics=xx:xx: ...   Changes the tree graphics characters. The values given are the hex codes for the following shapes:

```
        (1)       (2)   (3)      (4)     (5)

                   |             |        |
        -----      |    --+--    +--      +--
                   |      |      |
```

The values are two-digit hex codes separated by colons. For example, if the IBM extended character set is available, the following values may be used:

```
graphics=C4:B3:C2:C3:C0
```

nographics   Restores the default graphics characters.

head:symbol   Suppresses the full call tree and shows a call tree with the specified symbol as the top node. Multiple top nodes may be specified.

help   Outputs a help screen describing tree sub-options and terminates FortranLint.

{no}library   Shows calls made to routines defined in library template (**.lbt**) files. For additional information, see chapter 9.

{no}squish   Compresses call trees vertically by removing excess line graphics. The resulting trees are less readable, but require only half the space.

{no}trim	Trims the call tree by suppressing repeated subtrees. This is the default mode of operation. **notrim** may be used to disable trimming. **Note: "notrim" may produce call trees that require a large amount of disk space.**
{no}undefined	Shows calls made to routines that are undefined in the source code or libraries.

7.3 Call Tree Format

The call tree displays routines, subroutine calls, and function references in a graphical format. The starting routine is shown at the left top of the graph, and each level of routine calls is shown to the right of the calling routine. Each routine is connected to its called routines by lines drawn from dashes, vertical bars, and plus signs. Within each routine, calls are shown in the order they appear in the source code.

Routines that are not the program routine and are not called by any other routine are considered "detached". They will not appear in the main tree, but will be shown as the head of their own detached trees.

Symbol	Explanation
(*name*)	Parentheses are used to flag undefined routine *name*
(*n*)	Parentheses around a number *n* identify a trimmed subtree
{ *name* }	Braces are used to mark library routine *name* (from ".lbt" libraries)
[*name*]	Square brackets are used to mark Fortran 90 internal subprogram *name*
@*name*	Precedes calls to dummy routine *name*
* *name* *	Marks recursive chains that are chopped after the first iteration of *name*

7.3.1 Trimmed Trees

The size of call trees grows exponentially with program size. It's therefore impractical to generate complete call trees for large programs. As an alternative, FortranLint supports "trimmed" call trees.

In "trim" mode, FortranLint removes (or trims) duplicate subtrees. This brings tree size down to a reasonable level. At each "trim" point, FortranLint prints a subtree number that indicates where a master copy of the associated subtree can be found.

To enable "trim" mode, use the option switch "-**Ttrim**" or **/TREE=TRIM** (under VMS). The configuration file provided with FortranLint includes this option; FortranLint therefore uses "trim" mode by default.

Example: This is a "trimmed" tree (produced by **flint** -**Ttrim**):

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT (1)--PRINTIT-+-DIPSTAT--*PRINT*
        |                             |
        |                             +-GETUNIT
        |
        +-PRINT see 1
```

This is an "untrimmed" version of the same tree (produced by **flint** -**Tnotrim**):

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT--PRINTIT-+-DIPSTAT--*PRINT*
        |                         |
        |                         +-GETUNIT
        |
        +-PRINT--PRINTIT-+-DIPSTAT--*PRINT*
                         |
                         +-GETUNIT
```

7.3.2 Condensing Multiple Calls

By default, call trees show all of the calls made inside a given program. If one routine calls another several times, every call is displayed. As an alternative, FortranLint supports a "condensed" mode which shows the relationship between routines instead of the exact calling sequences used. To produce "condensed" trees, use the option **"-Tcondensed"** or **/TREE**=**CONDENSED** (under VMS). This option merges multiple calls from one routine to another into a single association.

Example: This is a "condensed" tree (produced by **flint** -**Ttrim,condensed**):

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE-+-PRINT (1)--PRINTIT-+-DIPSTAT--*PRINT*
        |         |                    |
        |         |                    +-GETUNIT
        |         |
        |         +-READNAME
        |
        +-PRINT see 1
```

This is an "uncondensed" version of the same tree (same **flint** command, omitting the **condensed** option):

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE-+-PRINT (1)-+-PRINTIT (2)-+-DIPSTAT--*PRINT*
        |         |           |             |
        |         |           |             +-GETUNIT
        |         |           |
        |         |           +-PRINTIT see 2
        |         |
        |         +-READNAME
        |
        +-READNAME
        |
        +-PRINT see 1
        |
        +-GETUNIT
```

7.3.3 Sorting Alphabetically

Calls are normally shown in order of appearance. To sort calls alphabetically (by routine name), use "-**Talphabetical,condensed**" or "/**TREE=ALPHA-BETICAL,CONDENSED**" (under VMS).

Example:

```
PROCDAT-+-GETUNIT
        |
        +-PRINT--PRINTIT-+-DIPSTAT--*PRINT*
        |                |
        |                +-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT--PRINTIT-+-DIPSTAT--*PRINT*
                                  |
                                  +-GETUNIT
```

7.3.4 Squished Trees

By default, FortranLint produces call trees that are double-spaced vertically. This improves readability. To produce single-spaced trees, use the option "-**Tsquish**" or /**TREE=SQUISH** (under VMS).

Note: Single-spaced trees are more compact. However, due to limitations of the ASCII character set, they are also harder to read. If an extended ASCII character set with line-drawing characters is available, the **graphics** option should be used in conjunction with **squish**. For additional information, see section 7.3.5.

Example: This is a "squished" tree
(produced by **flint -Tnotrim,squish,graphics= c4:b3:c2:c3:c0**):

```
PROCDAT──┬─GETUNIT
         ├─READNAME
         ├─SETTYPE──PRINT──PRINTIT──┬─DIPSTAT──*PRINT*
         │                          └─GETUNIT
         └─PRINT──PRINTIT──┬─DIPSTAT──*PRINT*
                           └─GETUNIT
```
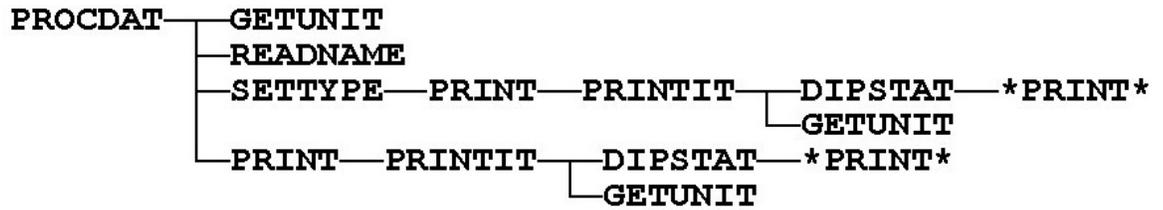
This is an "unsquished" version of the same tree (same **flint** command, omitting the **squish** and **graphics** options):

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT--PRINTIT-+-DIPSTAT--*PRINT*
        |                         |
        |                         +-GETUNIT
        |
        +-PRINT--PRINTIT-+-DIPSTAT--*PRINT*
                         |
                         +-GETUNIT
```
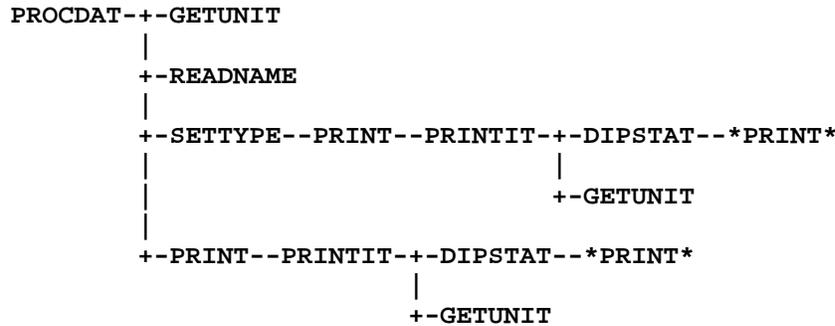
7.3.5 Graphic Character Set

By default, the call tree uses the ASCII characters "-", "|", and "+" to connect the routine names. To specify alternate characters, use:

      -Tgraphics=xx:xx:xx:xx:xx
or
      /TREE=(GRAPHICS=xx:xx:xx:xx:xx)   under VMS

where **xx** entries are ASCII character codes expressed as two-digit hexadecimal values. The five entries are interpreted as follows:

   (a) 1st code: horizontal connector
   (b) 2nd code: vertical connector
   (c) 3rd code: T intersection
   (d) 4th code: "|-" intersection
   (e) 5th code: L intersection

For example, if the IBM extended character set is available, use **c4:b3:c2:c3:c0**. Below is a "squished" tree
(produced by **flint -Tnotrim,squish,graphics= c4:b3:c2:c3:c0**):

```
PROCDAT──┬─GETUNIT
         ├─READNAME
         ├─SETTYPE──PRINT──PRINTIT──┬─DIPSTAT──*PRINT*
         │                          └─GETUNIT
         └─PRINT──PRINTIT──┬─DIPSTAT──*PRINT*
                           └─GETUNIT
```
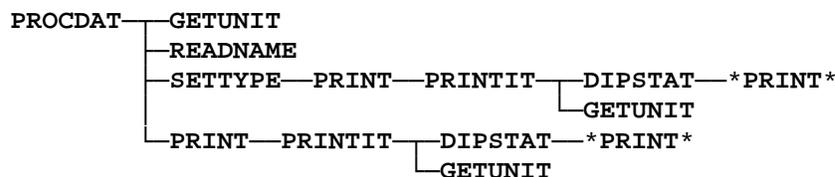
The default values are **2d:7c:2b:2b:2b**; see Section 7.2.1. To restore the default values, use "-**Tnographics**" (UNIX) or **/TREE=NOGRAPHICS** (VMS).

7.4 Call Tree Content

7.4.1 Top Node

Call trees can be generated with any routine as the top routine. When the top routine is selected, the full tree and detached trees are suppressed.

To generate a tree starting at the routine *name*, use the option switch "-**Thead:***name*" or **/TREE=HEAD:***name* (under VMS).

To display multiple trees, specify multiple routine names. For example:

> -**Thead:PRINT,head:SETTYPE**

or

> **/TREE=(HEAD:PRINT,HEAD:SETTYPE)**   under VMS

will show trees for both "PRINT" and "SETTYPE":

```
This is a primary tree starting at the program 'PRINT'

PRINT--PRINTIT-+-DIPSTAT--*PRINT*
               |
               +-GETUNIT


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

SETTYPE--PRINT--PRINTIT-+-DIPSTAT--*PRINT*
                        |
                        +-GETUNIT
```

To cancel a previously specified "-**Thead**" (or **/TREE=HEAD**) switch, add "-**Tnohead**" or **/TREE=NOHEAD** (under VMS) to the command line. This will restore the full call tree.

7.4.2 Undefined Routines

Call trees normally include all calls, whether or not the called routines are defined in the current input files. FortranLint uses parentheses to flag undefined routines.

To suppress calls to undefined routines, use "-**Tnoundef**" or **/TREE= NOUNDEF** (under VMS). If these options are used, call trees will be restricted to calls between routines defined in the current input files.

To restore the default mode of operation (e.g., if "-**Tnoundef**" was set in a configuration file), use "-**Tundefined**" or **/TREE=UNDEFINED** (under VMS).

7.4.3 Library Routines

If "library" (**.lbt**) files are specified on the command line, call trees will include calls to the associated library routines. FortranLint uses curly braces (**{}**) to flag library calls.

Calls to library routines will be displayed whether or not the **noundef** sub-option is used (see section 7.4.2). However, calls *between* library routines are not displayed, in either case.

To generate call trees which exclude library calls, use "-**Tnolib**" or ∕**TREE=NOLIB** (under VMS).

To restore the default mode of operation, use "-**Tlibrary**" or ∕**TREE= LIBRARY** (under VMS).

For additional information on library files, see chapter 9.

7.5 Recursion

FortranLint uses a pair of asterisks to flag recursive calls. For example, see *PRINT* in section 7.4.1.

7.6 Dummy Routines

FortranLint uses "@" characters to flag indirect calls; i.e., calls to a routine which are made indirectly through the argument list of another routine.

7.7 Entry Points

The ">" symbol in a call tree indicates that the call was made through an entry point. For example:

```
        |
+-ENTRPT>SUB1
        |
```

where ENTRPT is the entry point into subroutine or function SUB1.

7.8 Fortran 90 Internal Subprograms

Square brackets ([ ]) surrounding a routine name indicate that the routine is a Fortran 90 internal subprogram or a module subprogram:

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

8

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
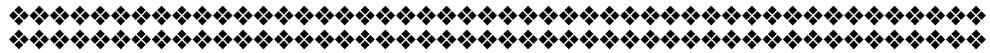
Cross Reference

8.1 Overview

To generate a symbol table cross-reference, use the option "-**x**" or **/XREF** (under VMS). For sample cross-reference output, see Appendix D or E.

Cross-reference tables can be generated from source files or from database files (see chapter 10).

The option switches "-**X**" and **/XREF** may be used to specify sub-options that control the format and content of the cross-reference table.

The UNIX cross-reference format/content sub-options are:

-Xfreeform	Free-form cross-reference
-X{no}tabular	Selects tabular format (vs. freeform)
-X{no}equiv	Selects equivalence usage information
-X{no}line	Line resolution (vs. subprogram resolution)
-X{no}legend	Selects legend for line resolution codes

The VMS cross-reference format/content sub-options are:

/XREF=freeform	Free-form cross-reference
/XREF={no}tabular	Tabular format (vs. freeform)
/XREF={no}equiv	Selects equivalence usage information
/XREF={no}line	Line resolution (vs. subprogram resolution)
/XREF={no}legend	Selects legend for line resolution codes

Two cross-reference formats are supported: **freeform** and **tabular**.

freeform is the default format. This format uses variable-length lines and shows information using a compact layout. The default sub-options for this format are "-**Xnoline**" and "-**Xnolegend**" or **/XREF=noline** and **/XREF= nolegend** (under VMS).

The **tabular** cross-reference format organizes fields into columns. This format is at least 132 characters wide. The default sub-options for **tabular** cross-references are "-**Xline**" and "-**Xlegend**" or **/XREF=line** and **/XREF=legend** (under VMS).

The **line** or **noline** sub-option sets the cross-reference to either line resolution or subprogram/program unit resolution, respectively. Subprogram/program unit resolution shows usage of a symbol within a subprogram/program unit, while line resolution shows usage of a symbol on each line in which that symbol appears. This must be set during source analysis to have effect.

If the "**-g**" (or **/GLOBAL)** option is used, the cross-reference will include additional information. Specifically, dummy argument usage is shown for subroutine and function definitions. In addition, the usage of the variables and arrays that are passed as actual arguments are determined.

8.2 Layout

Symbols are grouped into the following categories:

- Programs
- Block data subprograms/program units
- Subroutines
- Functions
- Modules (F90 only)
- Common blocks
- Structures
- Records
- Variables and arrays
- Parameters

Symbols are sorted alphabetically by name within each group.

If a symbol appears in more than one context (e.g., as a variable in one subprogram /program unit and as a subroutine name in another), the symbol is shown in both groups.

In the tabular format cross reference, the program, block data, module (F90 only), subroutine, and function sections are combined, as are the records and variable/array sections.

The information shown for each symbol will vary by category.

8.2.1 Program Routines

This symbol name is derived from the program name given on a program statement. If an unnamed program routine exists, it is given the name "Program". Multiple unnamed programs are named "Program2, Program3", etc.   The filename and the line number where the program routine begins and ends are shown along with the program name.

8.2.2 Block Data Routines

These are the symbol names from block data statements. Like program symbols, unnamed block data subprograms/program units are named "Blockdata", "Blockdata2", etc.

The filename and the line number where the block data subprogram/program unit begins are shown along with the block data name.

8.2.3 Subroutines and Functions

External procedures, internal procedures (F90 only), module procedures (F90 only), intrinsic procedures, and statement functions are shown in this section and are labeled correspondingly.

For functions, the data type is shown. This is normally the data type of the function definition. If the function is undefined, the data type of the first function call is used.

An internal subprogram (F90 only) has its parent routine as a qualifier using a double colon (::), for example, SUB**::SUB_INNER**.

If the code for the subroutine or function appeared in the sources analyzed, the filename and line number start/end of the subroutine/function statement are shown. If the definition was in a FortranLint library (**.lbt**) file, the name of the library is shown.

Argument descriptions of external, internal, and statement functions are also shown if the "**-g**" or **/GLOBAL** option was used during analysis. The argument descriptions show the class, data type, and usage of each argument. Argument class is one of:

<blank>	variable
array	variable or record array
subprogram	function or subroutine
return	alternate return
---	unused argument

Argument usage is indicated by the single-letter codes listed below:

Code	Description
S	set
R	referenced (used)
X	undetermined

For external and internal subroutines and functions, called routines are shown. If the table is in tabular format, the line number of each call is shown.

Finally, all calls to the function, subroutine or F90 module are listed. In the tabular cross-reference, the locations of the calls are shown by subprogram/program unit, filename, and line number in the References columns.

NOTE: to see all calls to/from the subprogram, add -**Mallcalls** to the Flint command.

8.2.4 Modules (F90 only)

These are the symbol names from module statements.

The filename and the line number where the module subprogram/program unit begins are shown along with the module name.

Modules referenced by this module are shown. If the table is in tabular format, the line number of each module reference is shown in the Calls column.

Finally, all references to the module via USE association are listed. In the tabular cross-reference, the locations of the calls are shown by subprogram/program unit, filename, and line number in the References column.

8.2.5 Common Blocks

Common blocks are shown along with their size (in bytes) and a list of their members. The routines that the common blocks appear in are shown, categorized into the following groups:

model	First instance of the common block. FLINT checks subsequent occurrences of the common block against this instance.
same	Matches the model.
names differ	Member types and sizes match the model, but they have different names.
layout differs	Member types and/or sizes don't match the model.

8.2.6 Structures and Structure Components

The cross-reference lists all structures used by the program, including their size, format, and members. Structures of the same name, size, and format are merged.

If the **linenumbers** or **tabular** format is selected, the cross-reference also includes a section labeled "Structure components" which lists occurrences of structure components. For additional information on the **linenumbers** and **tabular** formats, see section 8.3.

8.2.7 Variables, Arrays, and Records

Variables, arrays, and records are shown in this section. This includes automatic (local), dummy, common block members, and F90 module entities.

Arrays are distinguished by the dimension list. Each dimension is shown as either an upper bound or a lower/upper bound pair separated by a colon. If the lower or upper bounds are adjustable, "adj" appears. For open-ended dimensions, an asterisk (*) appears as the upper bound.

The Type column shows the data type including a length specifier for symbols or the name of the associated structure for records.

The Kind column shows the kind parameter of the symbol, if specified.

The Attributes column shows the attributes of each symbol. Attributes include **local, pointer, pointer based,** and **common block members**. Common block members are shown with the name and byte offset of the common block to which they belong.

The References column shows the cross-reference information for each symbol. The location resolution is either per subprogram/program unit or per line, depending on the setting of the "-**Xlinenumbers**" or **/XREF=linenumbers** option when the sources are analyzed. Subprogram/program unit resolution will show usage within each subprogram/program unit and is described in words. Line resolution will show usage for each line the symbol appears on, and its usage is described in single-letter codes.

Symbol usage is described as one or more of the following:

Line codes	Subprogram codes	Description
---	Unused	Symbol was not referenced, set, or indeterminate
A	Actual arg	Symbol passed as an actual argument
B	Array bound	Symbol was used as an adjustable bound for an array
C	Associated	Pointer has been associated with a target
c	Loop Counter	Symbol is used as a loop counter
D	---	Symbol appeared in a declaration (type decl, dim, common)
d	---	Symbol is implicitly declared
E	Equivalenced	Appeared in an equivalence statement
F	SF Dummy arg	Appeared as a statement function dummy argument
G	Ref as Label	An assigned goto jumped to label assigned to this symbol
i	Indirect Init	A symbol Equivalenced to this symbol was initialized
I	Initialized	Initialized in data statement, or when given data type
L	Set to Label	Symbol was assigned a label
M	Allocated	Symbol was allocated
N	Nullified	Symbol was nullified
O	Optl dummy arg	Symbol appeared as an optional dummy argument
P	Dummy arg	Symbol appeared in a subroutine or function statement
R	Ref	Symbol was referenced (its value was used)
r	Pointer ref	Target Symbol was referenced via its pointer
S	Set	Symbol was assigned a value
s	Pointer set	Target Symbol was set via its pointer
X	Indeterminate	May be ref or set, but exact usage cannot be determined
Z	Deallocated	Symbol was deallocated

Usage information (Ref/Set) is carried through all variable associations, including actual/dummy argument, common block member, and equivalence associations.

The default is to suppress unused common blocks. For additional information, see section 8.4.

8.2.8 Parameters

Parameters are shown along with their data type and their value. Parameters from different routines that have the same name and the same value will be merged.

The following usage codes apply to parameters:

Line codes	Subprogram codes	Description
D	---	Symbol appeared in a declaration (type decl, parameter)
R	Ref	Symbol was referenced (its value was used)
S	Set	Symbol was assigned a value (parameter statement)

The default is to suppress unreferenced parameters. For additional information, see section 8.4.

8.2.9 Equivalences

Entries for variables include equivalence information. For non-common block members, equivalences are named variables in the same scoping unit or the parent scoping unit. For common-block members, equivalences belong to the same common block.

Note: When a variable is equivalenced to an array element, FortranLint recognizes only the array name as equivalence. Consequently, when two variables that are not common block members are equivalenced to different elements of the same array, FortranLint will show the two variables and the array as the equivalence of one another. In case of two different scalar members of the same common block that are equivalenced to different array elements of the same array, FortranLint will show the array as the equivalence of both scalar members.

8.2.10 High Performance Fortran (HPF)

If HPF checking is enabled, and if the **linenumbers** or **tabular** format is selected, the cross-reference includes a section that displays occurrences of HPF processors and templates.

For additional information on HPF, see section 4.9. For additional information on the **linenumbers** and **tabular** formats, see section 8.3.

8.3 Format Selection

FortranLint allows users to select different formats for the cross-reference table by using the "-**X**" option (UNIX) or **/XREF** (VMS). Available formats include:

```
-Xfreeform              /XREF=freeform
-X{no}tabular           /XREF={no}tabular
-X{no}equiv             /XREF={no}equiv
-X{no}linenumbers       /XREF={no}linenumbers
-X{no}legend            /XREF={no}legend
```

A)   freeform / tabular

"-**Xtabular**" or **/XREF=tabular** (VMS) selects a table style that uses fixed-width columns. The output is 132 or more columns wide. The default line numbering mode for this format is "-**Xlinenumbers**" or **/XREF=linenumbers** (VMS).

"-**Xfreeform**" or **/XREF=freeform** (VMS) selects a more compact style with fields separated by single spaces. The default line numbering mode for this format is "-**Xnolinenum**" or **/XREF=nolinenum** (VMS).

B)   equiv / noequiv

By default, the cross-reference entry for a given variable includes usage information for the associated equivalences, whether or not the variable is used directly. To suppress equivalence info, use "-**Xnoequiv**" or **/XREF=noequiv** (VMS).

C)   linenumbers / nolinenum

To produce cross-reference tables with line numbers, use "-**Xlinenumbers**" or **/XREF=linenumbers** (VMS).

To limit cross-reference tables to the subprogram/program unit level, use "-**Xnolinenum**" or **/XREF=nolinenum** (VMS).

Note: To be effective, **linenumbers** or **nolinenum** must be specified *after* "-**Xtabular**", "-**Xfreeform**", **/XREF=tabular**, or **/XREF=freeform** on the command line.

D)   legend / nolegend

If **linenumbers** is selected, FortranLint prints single-character usage codes along with line numbers. A legend describing these usage codes is printed at the end.

To suppress the legend, use "-**Xnolegend**" or **/XREF=nolegend** (VMS).

To restore the legend (if it has been disabled), use "-**Xlegend**" or **/XREF=legend** (VMS).

Note: To be effective, **legend** or **nolegend** must be specified *after* "-**Xtabular**", "-**Xfreeform**", **/XREF=tabular**, or **/XREF=freeform** on the command line.

8.4 Content Selection

NOTE: As of 6.0, the separator between sentence fragments is now '.' instead of '_'. The '_' character is used in numerous intrinsic symbol names.

"-**X**" and ∕**XREF** both accept *content selection* arguments.

Content selection arguments are sentence fragments composed of one to six words, separated by periods or single dots. Each sentence fragment describes a criterion that can be used to select, add to, or filter cross-reference output.

A complete content selection includes the following words as its arguments:

-X{conjunction}{.usage}{.scope}{.class}{.named.xxx}

/XREF={conjunction}{.usage}{.scope}{.class}{.named.xxx} under VMS

Conjunctions	Usage (Adjective)	Scope (Adjective)	Class (Noun)
only	used/unused	local	routines
and	ref/unref	dummy/nondummy	programs
no	set/unset	statement/nonstatement	subroutines
	called/uncalled	intrinsic/nonintrinsic	functions
	indeterm/determ	global	blockdata
	actual/notactual	common/noncommon	modules
	init/uninit	external/nonexternal	extern
	decl/undecl	internal/noninternal	blocks
	equiv/unequiv		structures
	implicit/explicit		variables
	loopcounter/noloop		scalars
	associated/unassoc		arrays
			records
			parameters

Note: The default conjunction is "only".

While any of the words composing the criteria sentence are optional, the order of the words is significant. All words may be abbreviated, as long as they remain unambiguous. A few examples are:

Example	Result
-Xno.unused.variables	Suppress unused variables
-Xand.par.named.+oo	Also show parameters with names ending in "oo"
-Xonly.ref.dum.var.nam.i	Show referenced dummy variables named "i"
/XREF=common.arrays	Show arrays in common blocks
/XREF=arr.named.a?b+c	Show arrays named {any-letter}b{zero-or-more-letters}ca

(1)   The conjunction, if specified, must be first. This word specifies whether the criteria sentence is a selection, filter, or addition. The default mode of operation is "ONLY".

Word	Type	Description
ONLY	selection	suppress everything but the following (default)
NO	filter	suppress the following
AND	addition	add the following to what is already selected (will not be subject to previous filter criteria)

(2)   The usage adjective, if specified, must be next. This word acts on symbols at the subprogram/program unit level.

If a symbol is used in a particular subprogram/program unit in the fashion described by the usage adjective, the use of that symbol within the subprogram/program unit is included in the selection. If the usage adjective is omitted, FortranLint disregards usage when determining the selection.

Word	Antonym	Description
used	unused	Referenced, set, called, or indeterminate
referenced	unreferenced	
set	unset	
called	uncalled	
indeterminate	determinate	
actualarg	notactualarg	Variables passed to external routines
initialized	uninitialized	
declared	undeclared	Data type, dimensions, or common
equivalenced	unequivalenced	
implicit	explicit	
loopcounter	noloopcounter	
associated	unassociated	

Filtering acts on references at a subprogram/program unit level, filtering out the references to a symbol that match the filter criteria. If all references to a symbol are filtered out, the symbol itself is suppressed.

(3)   The scope adjective, if specified, must be next.

This is used in a similar fashion to the usage adjective but relates to the scope of the symbol. If the scope adjective is omitted, scope is not used in determining the selection.

Word	Antonym	Description
local	---	Dummy, Statement, Intrinsic, or Automatic
dummy	nondummy	Dummy argument
statement	nonstatement	Statement function
intrinsic	nonintrinsic	Intrinsic function
global	---	Common or External
common	noncommon	Common block or common block member
external	nonexternal	External routine

(4) The class noun, if specified, must be next. This specifies the class of the symbol.

The class noun describes categories of symbols. If the class noun is omitted, the selection contains all categories of symbols limited by the usage and scope adjectives.

Class	Subclasses	Description
routines		Includes programs, subroutines, etc.
	programs	
	subroutines	Includes dummy subroutines
	functions	Includes statement, dummy, and intrinsic functions
	blockdata	
	external	External routines which are undefined and unused
blocks	---	Common blocks
structures	---	
variables		Includes scalars, arrays, and records
	scalars	Single-valued variables
	arrays	
	records	Structured records
parameters	---	Defined in parameter statement

(5) The symbol name is specified last.

This is composed of two words, the word "named" followed by the actual symbol name. The following wildcards are allowed:

*   matches zero or more characters
?   matches one character

Wildcards may be combined.

Example:

    -Xnamed.ab*f?h
or
    /XREF=named.ab*f?h   under VMS

The default content of the cross-reference table is everything except unused common variables and unreferenced parameters.

UNIX examples:

-Xno.intrinsic	Suppress intrinsic functions
-Xuncalled.routines	Only routines that have not been called
-Xand.unused.parameters	Show unused parameters, too
-Xno.unused.common.variables	Don't show declarations of common variables where they are unused (if a symbol is never used, its name does not appear)

VMS examples:

/XREF=routines	Only show routine names
/XREF=no.common	Suppress common blocks and common block members
/XREF=unset.functions	Show only undefined functions
/XREF=unused.dummy.arrays	Show array dummy arguments that are unused

Multiple phrases may be given; these are checked in order from left to right. The phrases may be either on the same option or on separate options.

UNIX example:

-Xset.variables -Xno.unref.common.arrays -Xand.init.common.var

This will show set variables that are not unreferenced array common block members and any common block members that are initialized.

VMS example:

/XREF=routines,and.common.blocks

This will show routines and common blocks.

As mentioned previously, the default conjunction is "only".   The following combination will produce unexpected results:

-Xused.variables -Xequivalenced.variables

or

/XREF=used.variables /XREF=equivalenced.variables   under VMS

If the intent is to produce a cross-reference table with only variables that are used or equivalenced, the above command line option will not work. The second argument (equivalenced.variables) will override the first argument (used.variables), since the default conjunction is "only". The cross-reference table produced by this option will only include equivalenced variables. To perform the desired operation, use:

-Xused.variables -Xand.equivalenced.variables

or

/XREF=used.variables /XREF=and.equivalenced.variables   under VMS

Because Flint generates a cross reference (symbol table) with **all** symbols used throughout the program, a useful filter is to suppress local variables possibly being used as loop counters (e.g., i, j). To perform the desired operation, use:

-Xno.loop.named.?

9

❖❖❖
❖❖❖

Library Support

9.1 Overview

FortranLint supports a feature similar to 'C' prototypes. Specifically, the user can create "library shell" files for use in subsequent analysis operations.

NOTE: As of Flint version 6, the text-based .lsh files are preferred to the .lbt binary files. Section 9.3 is therefore deprecated.

A library shell (or **.lsh)** file is a text file that describes the interface structure of a library or package. **.lsh** files contain data structures similar to 'C' prototypes, but with additional information; specifically, reference/set flags and argument options.

Flint uses **.lsh** files to check calls to external libraries or packages. **.lsh** files allow Flint to perform interface checks even when library source code is unavailable. Reference/ set flags allow the user to describe individual routines more completely than traditional prototypes, improving the accuracy of generated reports. Argument-level options allow the user to "fine tune" interface checking for individual routines.

The Flint package includes several **.lsh** files, located in $FLINTHOME. As of version 7, these are:
- unixlib.lsh   Stubs for unix system routines
- vmslib.lsh   Stubs for VMS system routines
- openmp.lsh Stubs for OpenMP support routines
- ieeea.lsh   Stubs and definitions for IEEE Arithmetic
- ieeee.lsh   Stubs and definitions for IEEE Exceptions
- ieeef.lsh   Stubs and definitions for IEEE Features
- isobind.lsh   Stubs and definitions for ISO_C_BINDING
- isoenv.lsh   Stubs and definitions for ISO_FORTRAN_ENV
- MPI.LSH   Stubs for MPI subroutines and functions
- NetCDF.lsh Stubs for NetCDF subroutines and functions (F77 interface)

To use an **.lsh** file, simply add it to the project file list. If external routines are called, Flint will search the **.lsh** file for applicable definitions.

Note: Flint searches **unixlib.lsh** automatically for external routines, unless the routines are found in user-specified **.lsh** files. Under VMS, Flint searches **vmslib.lsh** instead. Also, the IEEE* and iso* **.lsh** files are included automatically when such module usage is detected.

For additional information about the search process, see section 9.4.

9.2 Writing Library Shell Files

Library shell files are simply text files containing Fortran subroutine or function stubs.

The FortranLint package includes two sample library shell files **unixlib.lsh** and **vmslib.lsh**. These files can be used to rebuild **unixlib.lbt** and **vmslib.lbt**, respectively (see section 9.3). They can also be used as the starting point for new library shell files.

Follow these guidelines:

(a) Use the filename extension .lsh.

(b) Write one or more Fortran subroutine or function stubs. The stubs should be named after corresponding library routines or system calls.

(c) Each stub should take the same arguments as the original routine, and should declare the arguments using the appropriate types.

(d) Function stubs should have the same return type as the original functions.

(e) Dummy arguments may be flagged with switches to provide additional information. (For additional information, see the following text.)

Example: This sample stub provides FortranLint with a description of the standard UNIX **exit** routine (treated as a subroutine):

```
subroutine exit (status)
integer status
end
```

This stub provides FortranLint with a description of the standard UNIX library routine **getcwd**:

```
integer function getcwd (dirname)
character*(*) dirname
end
```

As previously noted, arguments may be flagged with switches to provide additional information. For example:

```
subroutine exit (status/r)
integer status
end

integer function getcwd (dirname/s)
character*(*) dirname
end
```

The **/r** switch used here asserts that **exit** references the "status" argument. The **/s** switch used here asserts that **getcwd** sets the **dirname** argument. FortranLint takes this information into account when checking calls to these routines.

The following argument switches are supported:

/s   Asserts that the flagged argument is set.
/r   Asserts that the argument is referenced.
/i   Asserts that the argument's reference/set status is indeterminate.

These first three are the most common switches used. Example:

```
subroutine modtab (a/r, b/s, c/i)
real a, b, c
end
```

This stub asserts that "modtab" references its first argument and sets its second argument. The status of the third argument is indeterminate.

/l (Lower-case ell.) Asserts that the rest of the argument list (starting with the flagged argument) is option. For example:

```
integer function grade (name,class1/l,class2,class3)
character*40 name
integer class1,class2,class3
end
```

This stub asserts that the function "grade" takes one required argument (name), followed by zero to three optional arguments (class1, class2, and class3).

/o Asserts that the flagged argument is optional. For example:

```
subroutine foo (a, b/o, c/o, d/o, e)
integer a, b, c, d, e
end
```

This stub asserts that the middle three arguments to "foo" are optional.

/q Suppresses data-type checking and/or error #251 (scalar passed to array). Typically, this switch is used to flag arguments that can be represented in different ways.

For example, assume that a subroutine named "bar" takes a "quadword" (64 bit) argument, and assume that "bar" doesn't care if the caller passes a two-element array of **integer*4** or a four-element array of **integer*2**. In this case, the following stub could be used:

```
subroutine bar (x/q)
integer*2 x(4)
end
```

/v   Asserts that the flagged argument is passed by value. For example:

```
subroutine foo (n/v)
integer n
end
```

If this stub is used, FortranLint assumes that "foo" can be called as follows:

```
call foo(%val(3))
```

/z   Suppresses all interface checking for the flagged argument. For example:

```
integer function foo (a, b/z)
integer a, b
end
```

If this stub is used, FortranLint checks the first argument for every call to "foo", but does *no checking at all* on the second argument.

To combine two or more argument switches, use a single slash, followed by the appropriate letters. For example, **foo/or** asserts that the argument **foo** is optional, and that it is referenced.

Note: The **/s**, **/r** and/or **/i** switches cannot be combined for a given argument.

9.3 Creating Library Template Files – DEPRECATED

To create an "**.lbt**" file for a specific library or package, write an appropriate .lsh file (as explained in section 9.2), then execute a command similar to the following:

```
flint -L mylib.lbt mylib.lsh
```
or
```
flint /LIBRARY=mylib.lbt mylib.lsh                    under VMS
```

Substitute the appropriate name for "mylib".

This command will add entries from the library shell file **mylib.lsh** to the library template file **mylib.lbt**. If **mylib.lbt** doesn't exist, it will be created. Otherwise, the existing file will be updated.

Note: FortranLint can generate "**.lbt**" files directly from library sources; i.e., commands of the following form will work:

```
flint -L mylib.lbt *.for
```
or
```
flint /LIBRARY=mylib.lbt *.for                        under VMS
```

However, the "**-L**" and **/LIBRARY** options cause FortranLint to run in a special mode which bypasses normal analysis. Consequently, **input files must be debugged and free of errors before ".lbt" files are created.**

If library code is modified, repeat the original "-**L**" (or /**LIBRARY)** operation to update the "**.lbt**" file. Note that FortranLint will not update "**.lbt**" files automatically.

Example: To rebuild unixlib.lbt , go to the appropriate directory and execute:

   flint -L unixlib.lbt unixlib.lsh

To rebuild **vmslib.lbt** under VMS, go to the appropriate directory and execute:

   flint /LIBRARY=vmslib.lbt vmslib.lsh

9.4 Library Precedence

When analyzing calls to functions or subroutines, FortranLint uses the following definitions (highest precedence first):

(a) Definitions found in the user's Fortran source files

(b) Definitions found in the intrinsic table associated with the selected target compiler

(c) Definitions found in user-specified **.lsh** files

(d) Definitions found in unixlib.lsh or vmslib.lsh (under VMS)

By default, the intrinsic table takes precedence over user-specified **.lsh** files. To search user-specified **.lsh** files **before** the intrinsic table, specify the option "**-Muselbt**" or /**MISC=uselbt** (under VMS). If this option is used, (b) and (c) are reversed in the preceding list.

Note: The default system library templates **unixlib.lbt** and **vmslib.lbt** have the lowest precedence, whether or not the **uselbt** option is specified.

9.5 Miscellaneous Library Issues/ Usage Notes

9.5.1 Interaction with Cross Reference and Call Trees

Cross-reference tables and call trees automatically include referenced library routines. Cross-reference tables print library names along with routine names. Call trees use curly braces (**{}**) to flag library routines.

9.5.2 File Format

"**.lbt**" files are revision-locked. If incompatible "**.lbt**" files are used, FortranLint will print a warning message.

9.5.3 Preprocessor Directives

If your .lsh file contains preprocessor directives, change the extension from ".lsh" to ".LSH". This will automatically invoke the C preprocessor. Cleanscape-supplied MPI.LSH is named for just this reason.

9.5.4 Type-Declare Functions for which there is no Source

If your program uses functions from a 3rd party library or for any reason you do not have source code, you may get messages about type mismatches since Flint uses implicit typing if the function is not declared otherwise.

You can put these into a .lsh file, but there could be double-declare issues later if the source code containing the function is added to the Flint analysis later. However, that would be easy to correct.

An alternative is to simply declare the function inside the subprogram where it's being used. For instance, Intel's SYSTEMQQ function returns LOGICAL; if used in your code, you could simply add

```
LOGICAL :: SYSTEMQQ
```

and Flint would *not* assume it is REAL, and instead issue messages if it were used in a non-LOGICAL manner.

10

❖❖
❖❖

Database Files

10.1 Overview

FortranLint can be used to create *database* (or "**.fdb**") files for use in subsequent analysis operations.

A database (or "**.fdb**") file is a binary file that contains symbolic information obtained from one or more FORTRAN source files.

Files with the extension "**.fdb**" are database files generated by FortranLint during source code analysis. "**.fdb**" database files contain symbolic information for the modules processed.

"**.fdb**" files may be used to re-generate cross-reference tables, call trees, diagnostic messages, etc., without re-analysis of the original source code.

As of rev. 4.33B, "**.fdb**" files may also be used as libraries. In other words, "**.fdb**" files can be used instead of "**.lbt**" files on the command line. For additional information, see section 10.4.

10.2 Creating Database Files

To create database files, use the "**-B**" option or **/DATABASE** (under VMS) as follows:

>   flint -B*dbfile* foo1.f foo2.f foo3.f ...
> or
>   flint /DATABASE=*dbfile* foo1.for foo2.for foo3.for ...
>
>   under VMS

dbfile specifies the base name that should be used for the database. The filename extension "**.fdb**" will be added automatically.

If the specified database file already exists, it will be overwritten.

To suppress console error messages during database creation, add the option switch "**-Oall**" or **/SUPPRESS**=**ALL** (under VMS) to the FortranLint command line.

10.3 Using Database Files

To extract information from an existing database file, use a normal **flint** command with the database file as an argument. For example:

    flint -t *dbfile*.fdb
or
    flint /TREE *dbfile*.fdb   under VMS

The command line should not specify any other database files or FORTRAN source files.

All call tree options are available when database files are processed. Most of the cross-reference options are available; **line** is an exception. "-**Xline**" and **/XREF=line** are ignored; **flint** uses the **line** value set when the database file was generated.

Additionally, if "unreferenced parameters" or "unused common block members" are not selected when the database file is created, the associated messages will not be provided by subsequent database queries.

 "**.fdb**" files should be regenerated whenever the associated source code is modified.

10.4 Using FDB files as libraries.

For FLINT rev. 4.33B or later, "**.fdb**" files can be used as libraries. In other words, you can specify "**.fdb**" files instead of "**.lbt**" files on the command line.

There is one special case: If the file list starts with an "**.fdb**" file, FLINT runs in "database" mode, and all other file arguments are ignored. For more information about "database" mode, see section 10.3.

Two option switches may be used to control the way "**.fdb**" libraries are used:

1. **–Mlibcom**. By default, FLINT doesn't check source-level common blocks against common blocks declared inside "**.fdb**" libraries. If "-Mlibcom" is used, FLINT checks source-level commons against all "**.fdb**" files specified on the command line. "-Mlibcom" also suppresses not-referenced/not-set messages for commons in the user's code, which are referenced or set at the "**.fdb**" level.

2. **–Mlibext**. By default, FLINT searches all specified "**.fdb**" files for missing procedures. If "-Mlibext" is used, searching is suppressed; unresolved procedures are treated as external, whether or not they are defined inside "**.fdb**" files.

Restriction:

FLINT does not yet support translation of "library shell" (**.lsh**) files to "**.fdb**" format. (For information on .lsh files, see chapter 9.)

11

❖❖
❖❖

Xlint Introduction

NOTE: Xlint has been deprecated on **most** hosts and is **not available** under Windows. A GUI interface is provided for most host platforms. See the flintguide.pdf manual in the 'doc' subdirectory for more information.

Therefore, Windows users can skip Chapters 11-16.

Xlint is a Motif-based programming tool. It is designed to provide FORTRAN developers with an interactive graphical user interface that can be used to browse FORTRAN source files.

Xlint operates on the database (or "**.fdb**") files generated by FortranLint. With four windows displaying information, Xlint allows the developer to step through potential errors and to see the relationships between source code, call tree and symbol table information, all on one screen.

Symbol cross-reference information can be automatically brought up, showing all code references to any symbol in the analyzed program. Each occurrence of a symbol can quickly be found in the source code; at the same time, the appropriate node is highlighted in the displayed call tree.

12

❖❖
❖❖

Learning About Xlint

This chapter will cover the basics that users need to know before running Xlint and will help them understand the usage and function of the many options available within this product.

12.1 Screen Layout

The Xlint screen is made up of four windows and a control panel. See Figure 12-1.

The four windows from top to bottom are **Source**, **Lint**, **Tree**, and **Cross Reference**. They are used to display specific information relating to the current database being analyzed. When information in any of the windows exceeds the size of the window, a scroll bar will appear on the bottom and/or right-hand side. Each window can be enlarged or reduced at the expense or benefit of the other windows.

The **Control Panel** bar between the Source and Lint windows contains a text input field and three buttons.

Popup menus are supported by all four windows and can be called up by a click of the right mouse button, with the pointer anywhere in the appropriate window. Options in the popup menus can be selected by simply holding down the right button, dragging the pointer to the desired option, and then releasing the button. When the button is released, the option will be set and the menu will disappear.

On-screen help is available. A help menu can be called up by a click of the left mouse button on the **help** option at the top of the Xlint screen.
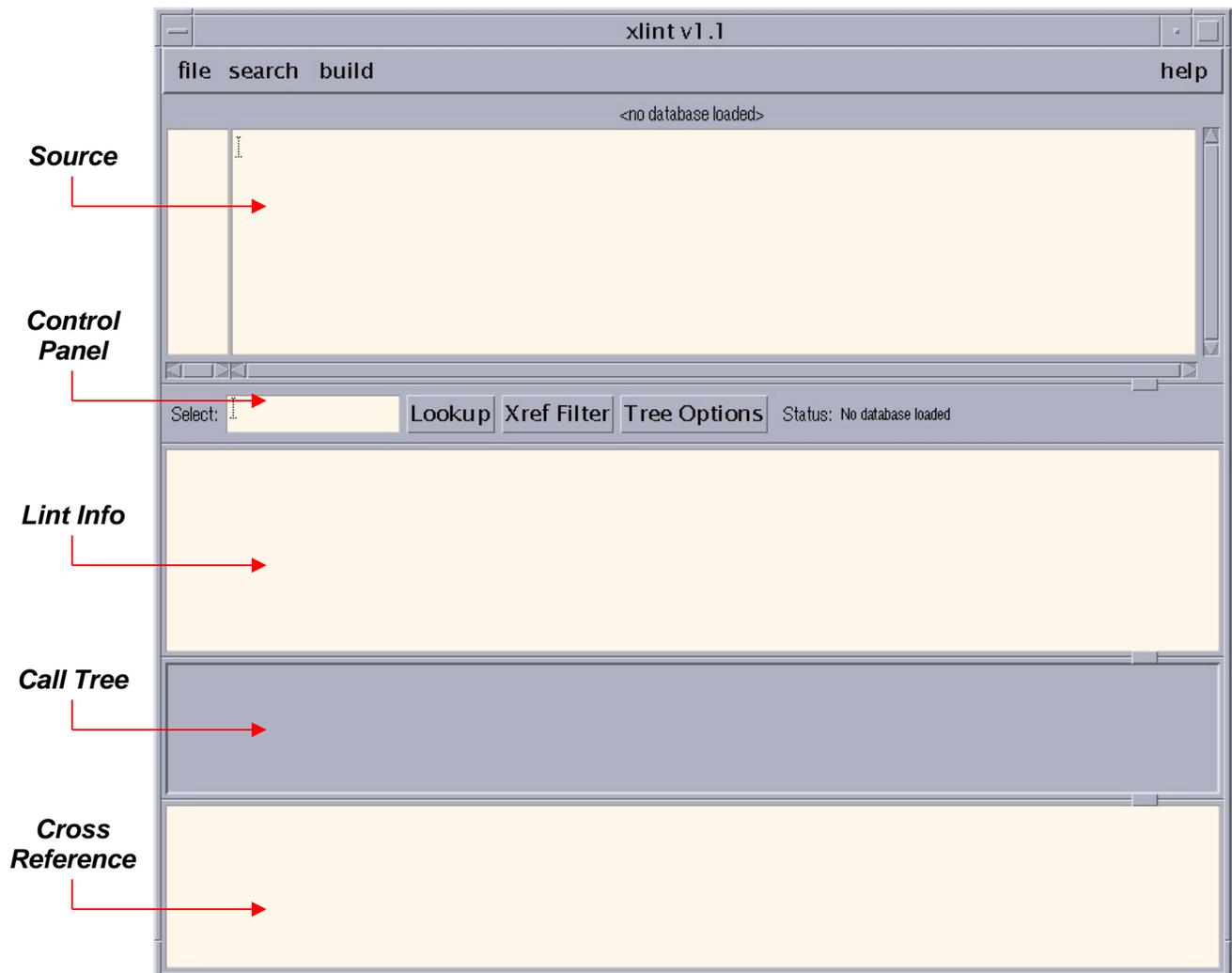
Fig. 12-1: Xlint window (on startup)

12.2 File Menu

The File menu is used to select the database or source file.

A submenu with five options can be brought up by a click of the left mouse button.

Load Database   Used to load a pre-existing database. A database must be loaded before symbol information can be displayed in the windows.

View File   Used to load an arbitrary file into the Source window. It will stay loaded until an action from another window calls up a different source file.

Save File   Used to save any edits made to the file in the Source window. Editing must be enabled first.

Enable Editing   Allows the file in the Source window to be modified and saved. Changes are only saved when the Save File menu item is selected. When the Source window is in the editing mode, the **Disable Editing** option replaces Enable Editing in the submenu and disallows any editing in the Source window.

Quit   Exits Xlint.

12.3 Search Menu

The Search menu allows text searches on the Source window, using the currently highlighted text as the search string.

Previous   Searches backwards for the selected text.

Next   Searches forwards for the selected text.

Go To Line   Uses the selected text as the line number and goes to that line. For example, if the highlighted text is the number "27", then if line 27 exists, the program cursor moves to the 27th line.

12.4 Build Menu

The Build menu is used to create or update a database (**.fdb)** file.

The status field on the control bar shows the status of the most recent rebuild.

Configure	Used to select the name of the database, the source files, and options used during source processing.
Rebuild	Runs FortranLint with the configured options and files to regenerate the database file.
Kill Rebuild Process	Stops the source processing; available only during rebuild.
Use Rebuilt Database	Loads the database that was last rebuilt by the Rebuild menu selection. This has the same effect as loading the database from within the File menu.
View Build Output	Pops up a window that shows the output from Fortran-Lint during the rebuild process. The FortranLint output can be used to determine why a "build" operation failed.

12.5 Source Window

The source code currently being analyzed is displayed in this window.

Action in other windows will cause files to load automatically and jump to the appropriate point in the source. A lookup can be performed on highlighted items that can affect the Tree and Cross Reference windows.

The popup menu can be used to select the source related to what is currently highlighted in the other windows. With these options, users can step through the cross-reference entries one by one or repeat a text search with a click of the mouse.

Previous Xref	Goes to the text referred by the cross-reference entry one before the current highlight in the Cross Reference window.
Current Xref	Goes to the text referred by the cross-reference entry currently highlighted in the Cross Reference window.
Next Xref	Goes to the text referred by the cross-reference entry one after the current highlight in the Cross Reference window.
Previous Text	Goes to the previous occurrence of the text currently highlighted in the Source window.
Next Text	Goes to the next occurrence of the text currently highlighted in the Source window.

Lint	Goes to the text referred by the lint message currently highlighted in the Lint window.
	Note: If the Lint window is currently in the "summary" mode, the cursor in the Source window will not be affected.
Tree	Goes to the function or subroutine currently highlighted in the Tree window.

12.6 Lint Window

The current FortranLint analysis messages pre-generated by FortranLint are displayed in this window.

When a database is initially loaded, a summary of the FortranLint source analysis is shown. Double-clicking an item in the summary calls up the actual instances of that message. Double-clicking a message instance causes lookups in the other windows specified for action in Lint's popup menu.

Summary	Shows a summary of FortranLint analysis output.
	Example:
	IMPLCT #125 7x: symbols were implicitly typed as *: *
	where **7x** means that there are 7 instances of such message.
All Messages	Shows all the actual message instances of the FortranLint source analysis. Each message includes the line number, the subroutine it belongs to, the message number, and the message itself.
	Example:
	demo.f(33)[PRINT] #125: symbols were implicitly typed as I*4: IUNIT
Action	The Action popup menu determines which of the other windows are influenced by actions taken in the Lint window.
	If a message is double-clicked in the Lint window, depending on the selections in the Action popup menu, one or more of the other windows is changed to reflect the new selection. The Tree window will change to reflect the routine where the message was reported. The Cross Reference window will change to reflect symbol information associated with the message. By default, both Cross Reference and Tree are selected.

12.7 Tree Window

This window contains a graphical representation of the program's "call" structure, centered around a given routine.

Each node of the call tree represents a routine. The selected routine is placed in the center. The routines to the left and right are the predecessors and descendants of the selected routine, respectively.

Clicking a node will highlight it and make it selectable from the popup menus of the other windows. Double-clicking a node will re-center the tree around that node. Double-clicking a node while holding the Shift key (<Shift>double-click) will cause lookups in the other windows specified for action in the Tree's popup menu.

The popup menu in the Tree window allows the tree root to be set from the current routine in the Source, Lint, or Cross Reference window.

Selected Routine	Uses the currently highlighted routine name from the Source window or the text input field.
Routine Containing Lint	Redisplays the Tree window with a tree centered around the routine containing the current lint message.
Routine Containing Xref	Redisplays the Tree window with a tree centered around the routine containing the current Cross Reference entry.
Action	Selects the affected windows when a shift double-click is done on a tree routine. By default, the Source and Lint windows will reflect the change on the Tree window.

12.8 Cross Reference Window

The Cross Reference window contains a cross-reference for the selected symbol.

Symbols may be selected by name and may contain wildcard characters. The cross-reference entries are filtered by the settings in the Xref Filter selection box in the Control Panel (see next section).

The following wildcard characters are accepted:

*	**zero or more characters**
?	**any character**

Double-clicking a cross reference entry calls up the source and/or call tree related to that entry, depending on the action settings.

Options available in the popup menu are as follows.

Lookup Selected Symbol

Looks up the symbol currently highlighted in the source window.

Lookup Tree Routine

Allows users to look up the cross-reference messages related to the routine currently highlighted in the Tree window.

Lint References   Shows cross-reference information regarding the lint message highlighted in the Lint window.

Action   Determines the affected windows when a double-click is done on a cross-reference entry.

12.9 Control Panel

The control panel bar between the Source and Lint windows has four labels that will perform various functions. A description of each option follows:

Select   Allows the user to type in a symbol in the field next to the Select option for lookup.

To perform the lookup, the user can either hit return at the end of the text or click on the **Lookup** button.

Lookup   Uses the highlighted text from the Source window to affect the windows selected for action by the Source window's popup menu.

If no text is highlighted, the text specified in the Select field to the left will be used. If text is entered into this Select field and <Return> is pressed, the entered text is used regardless of what is highlighted in the Source window.

Xref Filter   Calls up a selection box to select the types of symbols to show in the Cross Reference window. Any number of qualifiers may be selected.

Tree   Brings up a selection box to set the "parent" and "child" depths of the call tree to be displayed in the Tree window, as well as condensing multiple calls.

Parent depth is the number of levels shown upward in the call stack in relation to the selected tree routine. **Child** depth is similar, but in a downward direction. The toggle for **Condense** mode causes multiple calls from one routine to another to be shown as one link, rather than duplicated. **Library** includes the library routines defined in ".**lbt**" files. **Undefined** shows all routines which are called, whether or not they are defined in the current input files.

12.10 Mouse Functions

The functions of the mouse are consistent with standard Motif usage.

Left button   Used to select menu options and buttons; can also be used to highlight or mark text in the Source window. Highlighting is accomplished by pointing to the beginning of the text the user wishes to mark and, while holding the left button down, dragging to the end of the text and then releasing the left button.

Middle button   Used to paste highlighted text at the current cursor location or text prompt. Pasting or inserting cannot be done into the Source window.

Right button   Used to call up the popup menus and select the options in these menus.

13

❖❖
❖❖

Database Files and Xlint

13.1 Overview

As explained in chapter 10, database (or "**.fdb**") files are binary files that contain symbolic information for one or more FORTRAN source files.

Xlint uses the information stored in "**.fdb**" files to browse the associated source files and/or analysis output.

Database files may be created from the command line (using FortranLint), or they may be generated inside Xlint. For the command-line procedure, see section 10.2. Section 13.3 covers the Xlint procedure.

Note: This chapter assumes that the environment variable (or VMS logical) **XLINTPATH** is set properly. If Xlint is being run from a directory other than the directory that contains project sources, **XLINTPATH** should point to the directory that contains the sources.

13.2 Loading Database Files

Before Xlint can be used, a database ("**.fdb**") file must be loaded. To load a database file inside Xlint, proceed as follows:

1)   Select the **File** menu from the options at the top of the screen.

2)   Select **Load Database** from the File menu. This will bring up a directory and file selection screen.

3)   Select the appropriate directory. To select the desired "**.fdb**" file, double-click the file *or* click the file to highlight it and then click **OK**.

The "lint" summary for the specified "**.fdb**" file should appear in the Lint window.

Alternatively, a database file may be specified on the Xlint command line. For additional information, see section 15.3.

13.3 Rebuilding Database Files under Xlint

The **Build** menu on top of the Source window can be used to rebuild an existing database (or to create a new one). The "build" procedure is as follows:

1)   Select **Configure** on the Build menu. Enter the database name (without the ".**fdb**" extension) in the **Database** field. Enter the associated source file names in the **Source Files** field. Set the other options as desired. See figure 13-1.

2)   Select **Rebuild** on the Build menu. The **Status** field in the Control Panel may be used to monitor "build" status.

3)   When the Status field shows "Rebuilding completed", the new database may be loaded. (To do this, select **Use Rebuilt Database** on the Build menu.)

Users may also see the FortranLint output by using the **View Build Output** on the Build menu. If the "build" failed, **View Build Output** can be used to determine the cause. (For additional information, see section 12.4.)
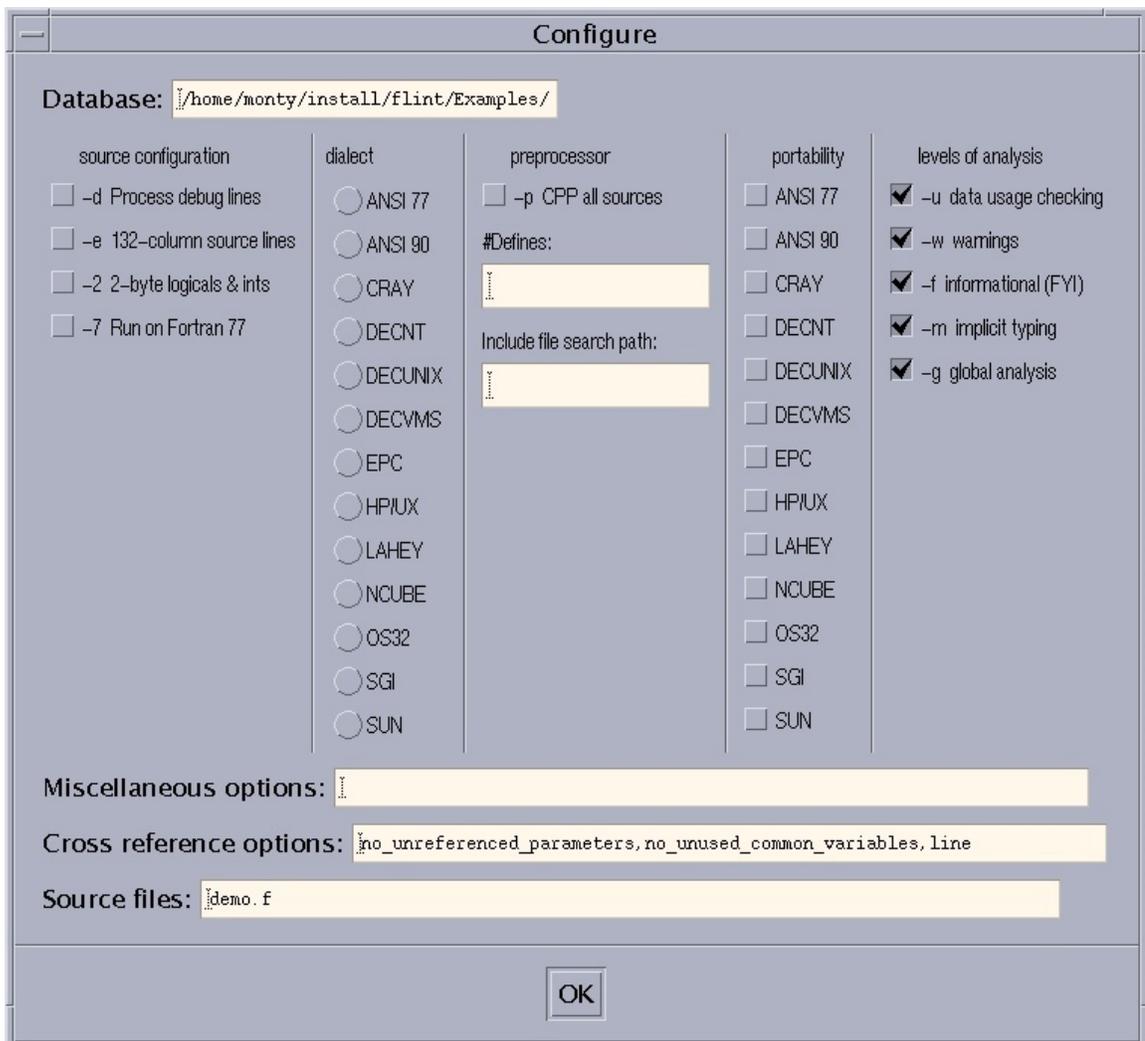
Fig. 13-1: Xlint Build-Configuration window

14

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Xlint: Getting Started

14.1 Configuration Setup

For installation instructions, see Appendix H or I.

In particular, note that a "resource file" should be copied to the appropriate directory. (For additional information on resource files, see the installation instructions and chapter 16.)

Also note that three UNIX environment variables (or VMS logicals) should be set for each Xlint user:

XLINTHOME	Path for the directory that contains the Xlint support files.
XLINTHOST	Network name (or node name) for the system running the Xlint license manager.
XLINTPATH	Path for the directory that contains the user's FORTRAN source files.

14.2 Running Xlint

Before Xlint can be used, the user must create a project database (or "**.fdb**" file). For additional information, see chapters 10 and 13.

To run the browser, enter the command **xlint**:

    xlint

The Xlint menu will appear, along with four empty windows.

Next, use **Load Database** on the File sub-menu to load the appropriate database. (For additional information, see section 13.2.)

After the "**.fdb**" file is loaded, source-analysis output will appear in the Lint window.

To display all lint messages, scroll through the lint message summary. To display all occurrences of a given message, double-click the message. The Lint window will be updated appropriately.

For detailed information on a given occurrence, double-click the occurrence. Xlint will display related source code, call tree output, and cross-reference information.

For on-screen help, click the left mouse button on the **Help** field in the upper right corner of the Xlint screen.

To load another database, select **Load Database** on the File menu.

To exit Xlint, select **Quit** on the File menu.

14.3 Sample Sessions

The FortranLint / Xlint package includes sample FORTRAN 77 and Fortran 90 project files.

The sample FORTRAN 77 files include **demo.for** (an F77 source file) and **demo.fdb** (the associated database file). **demo.for** may be used to rebuild **demo.fdb**; for additional information, see section 13.2.

Similarly, the sample Fortran 90 files include **demo90.for** (an F90 source file) and **demo90.fdb** (the associated database file). **demo90.for** may be used to rebuild **demo90.fdb**.

Session 1:

1)   Under the FortranLint / Xlint installation directory, run:

```
xlint  demo.fdb
```

This will bring up the Xlint menu and analysis output for **demo.fdb**.

2)   To find all symbols starting with "I", enter "I*" in the **Select** field on the Control Panel and press <Return>.

This will bring up cross-reference information for all variables beginning with the letter "I".

3)   To bring up information related to a specific symbol, double-click the appropriate line in the Cross Reference window. For example, to display information related to the variable "INUIT", double-click the following line:

```
INUIT,I*4 variable,in demo.f(43)[PRINTIT] is Set,Actual arg
```

The Source window should display **demo.for** with line 43 (containing INUIT) highlighted. The Tree window should display a tree centered around the PRINTIT routine. The screen should appear similar to that shown in Figure 14-1.

4)   Double-click another cross-reference entry. For example:

```
I, I*4 variable, in demo.f(6)[PROCDAT] is Ref
```

Note the changes in the Source and Tree windows.

<u>Session 2</u>:

1)   Run:
            xlint

This will bring up the Xlint menu with four empty windows.

2)   Click on **File** in the upper left corner of the screen. Select **Load Database** to bring up the **Load Database** dialog box. Use the **Directories** and **Filter** options to go to the FortranLint / Xlint installation directory. Double-click **demo.fdb** to select it *or* single-click to highlight **demo.fdb** and press **OK.**

Xlint should display analysis output for "demo.for" in the Lint window.

3)   To display all the occurrences of a particular lint message, double-click the message with the left mouse button.

For example, double-click the following message:

```
IMPLCT #125 7x: symbols were implicitly typed as *: *
```

to display all occurrences of IMPLCT #125. Note that **7x** means that there are 7 instances of this message.

4)   After all instances are displayed, double-clicking one of the instances will display information related to the instance.

For example, double-click the following instance:

```
demo.f(33)[PRINT] #125: symbols were implicitly typed as I*4:
IUNIT
```

The Source window should display source code with the highlight on IUNIT in line 33. The Tree window should display a call tree centered around the PRINT routine. In the Cross Reference window, the following line should be highlighted:

```
IUNIT, I*4 variable, in demo.f(33)[PRINT] is Dummy arg
```

5)   Double-click a lint instance that is not currently highlighted. For example:

```
demo.f(49)[DIPSTAT] #125: symbols were implicitly typed as
R*4: PRINT
```

The Source window now redisplays the source file with the highlight on
"PRINT" in line 49. The Tree window shows a tree centered around
DIPSTAT. The Cross Reference window shows the following information:

```
PRINT, subroutine, in demo.f(49)[DIPSTAT] is called
```

6)   To see the lint summary again, use the right mouse button to select **Summary**
in the Lint window.
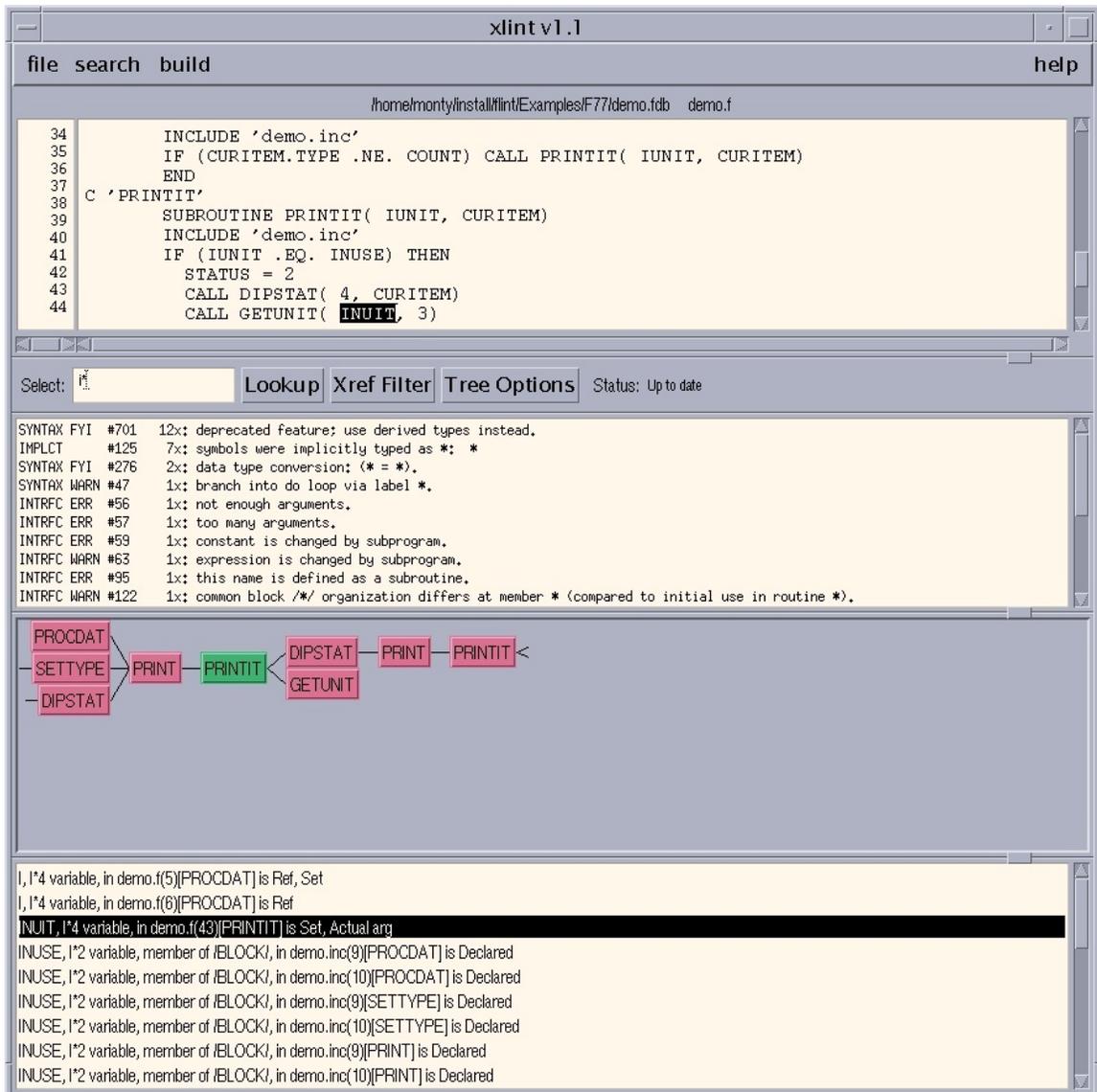
Fig. 14-1: Xlint Window for Sample Sessions 1 and 2

15

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

More About Xlint

15.1 Resizing Windows

Any of the Source, Lint, Tree, or Cross Reference windows in the Xlint screen can be enlarged or reduced at the expense or benefit of the other windows. To resize a window, press the left or middle mouse button on the small box between two windows, and drag it to the new boundary line as desired. Then release the button.

When the information in any of the windows exceeds the size of the window, a scroll bar will appear on the bottom and/or right-hand side of the window.

15.2 Window Interaction

Window interaction is controlled by the **Action** sub-menu in each of the Lint, Tree, and Cross Reference windows. The settings in these **Action** menus determine how changes in that particular window will affect the other windows. Depending on the action settings, all the windows may be updated to reflect information relative to the changed window.

To view or change the action settings for a particular window, move the mouse cursor to any location within that window and press the right mouse button. The popup menu for that window will now be displayed. Move the mouse to the **Action** option and press the right button again. The action options will now be displayed. To toggle an option on or off, simply point to that option box, and press the right mouse button.

The default settings for window interaction are set so that an action in any window will affect the others. For example, if the user double-clicks a cross-reference entry, the corresponding source code and the tree information will appear in the Source and Tree windows.

15.3 Command-Line Options

Xlint supports the standard "X" command-line options (i.e., -bg, -fg, -display, etc...). For additional information on these options, see the system vendor's "X" documentation.

Additionally, a database ("**.fdb**") file may be specified on the command line. For example:

>   xlint foo.fdb

To specify an alternate resource file (e.g., **bar.dat**), use an option of the following form:

>   xlint -rf bar.dat

This option loads both the default resource file and the user-specified file. Options in the user-specified file take precedence.

15.4 Advanced Example

The following session will use "demo.fdb" as the example. A screenshot of the sample session is shown in Figure 15-1.

<u>Sample session 3</u>:

1)   In the FortranLint ∕ Xlint installation directory, run:

>   xlint demo.fdb

This will bring up an Xlint screen with analysis output for the "demo" project in the Lint window.

2)   Click the left mouse button to bring up the File menu. Select the **View File** command. Select "demo.for" (or "demo.f"). The Source window should display FORTRAN source code for the "demo" project.

3)   Symbols can be located throughout the loaded source file by highlighting the text of a symbol in the Source window, clicking the right mouse button to bring up the popup menu, and selecting **Next Text**. For instance, highlight "PRINT" in line 9 in the Source window.

To search for the next "PRINT" in "demo.for", click the right mouse button anywhere in the Source window to call up the popup menu. Select **Next Text**.

The next "PRINT" highlighted in the Source window is in the SETTYPE subroutine.

4)   To see all occurrences of "PRINT", use the right mouse button in the Cross Reference window to call up the submenu. Choose **Lookup Selected Symbol**.

5)   To make a cross-reference entry available as a selection for the Source and∕or Tree windows, highlight the entry using a click of the left mouse button. For example:

PRINT, subroutine, in demo.f(33)[PRINT] is defined

6)   To see the tree related to this highlighted cross-reference entry, use the right mouse button to select the **Routine Containing Xref** in the Tree's popup menu.

      The Tree window shows a call tree centered around the PRINT routine.

7)   To see where in the source code the highlighted "PRINT" in the Cross Reference window refers to, use the right mouse button to select "**Current Xref**" in the Source window.

      The highlight in the Source window now moves to the symbol "PRINT" in line 33.

      Note that the combined result of steps 5-7 can be done by simply double-clicking the cross-reference entry in step 5.

8)   <Shift> double-click the SETTYPE routine in the Tree window. The Source window moves the highlight to "SETTYPE" in line 27. The Cross Reference window shows all the cross-reference information about SETTYPE.

Users can change the setting for any window at any time. For example, the depth of the parent or child tree in the Tree window can be changed by using the **Tree Option** in the Control Panel. The tree will be redisplayed with the new depth.

Fig. 15-1: Xlint Window for Sample Session 3

16

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Xlint Resource Files

16.1 Overview

A *resource file* is an ASCII text file that contains the configuration information needed for Xlint to run. Xlint resource files conform to the standard X Window resource file conventions.

The default resource file is named **Xlint** or **XLINT.DAT** (under VMS). It is strongly suggested that **the original copy of this resource file not be altered**. If users need to modify the default configuration, they should create modified copies.

Users may load modified versions of the resource file in various ways; for additional information, see section 16.2.

16.2 Xlint and XLINT.DAT

The Xlint resource file is named **Xlint** or **XLINT.DAT** (under VMS).

A copy of this file should be placed in the home directory for each Xlint user. By default, Xlint uses this copy. Users may specify alternate versions on the Xlint command line; for additional information, see section 15.3.

Alternatively, , users may set the standard environment variable **XAPPLRESDIR** or use the standard **app-defaults** directory.

Under VMS, two logicals **DECW$SYSTEM_DEFAULTS** and **DECW$USER_DEFAULTS** are used. To install a copy of **XLINT.DAT** for system-wide use, place it in the directory specified by **DECW$SYSTEM_DEFAULTS**. To install a copy of **XLINT.DAT** for use by an individual user, place it in the directory specified by **DECW$USER_DEFAULTS** for that user.

If Xlint finds copies of the resource file in two or more places, all of the specified options are used, but options in individual user resource files take precedence over options in system-wide resource files.

If a resource file is specified on the Xlint command line, options in the specified file take precedence. For additional information, see section 15.3.

Appendices

Appendix A

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Installation Windows, Unix/Linux

A.0 Windows Installation – GUI only

1. Installation.

(a) Execute `flintgui<ver>_win.exe`. Note: You can run this program from any drive or directory. Note that there is no command-line-only installer, but Flint can be run from the command line with appropriate settings.

An installer window should appear. Click the Install button. This should extract a number of files. After all files are extracted, click Exit.

As of Flint version 6, the installer will configure your machine with the demo version of Flint. The demo version requires no key and is thus suitable for evaluation customers. It is fully functional, as can be evidenced by reviewing the summary listing it provides. However, only a limited number of detailed messages, pinpointing the error to the line of code in the source, are provided.

To bypass demo installation, click No when the dialog box appears. Note that a key will be required! To change at any time, simply copy flintfull.exe to flint.exe, or flintdemo.exe to flint.exe, depending on which version you want.

The installer creates a shortcut on your Desktop. The shortcut should appear roughly one second after all files are extracted.

(b) Run FortranLint. To do so, double-click the FortranLint shortcut.

(c) If you install FortranLint under Windows 2000 or later as Administrator, and you want to make the program accessible to ordinary users, some additional steps are required. For more information, see flintguide.pdf, located in your 'doc' subdirectory).

The GUI version of FortranLint is now installed. Note: A reboot is not required. To run the program, use the shortcut created by the installer.

To more details, please see section 2.1 of flintguide.pdf.

2. Command-line setup.

If you'd like to use the command-line version of FortranLint, follow the steps outlined in Part 1, then two additional steps are required. These are done in the Environment Variables section of System Properties, or may be done in an existing command session as follows:

(1)   Set the environment variable FLINTHOME:

    SET FLINTHOME=C:\CLEANSCAPE\FLINT\MAIN

 (2) Modify PATH as follows:

    PATH %FLINTHOME%;%PATH%

    Note: Under Windows 98, you may need to add double quotes as follows:

    PATH "%FLINTHOME%;%PATH%"

If you have questions, contact support@cleanscape.net.

3. License Manager.

If you are a demo user, there is no license management.

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

The Windows version uses the Reprise RLM License Manager. 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.

4. Documentation.

For more information, see the following two documents:

flintman.pdf	Main FortranLint manual	(PDF format)
flintguide.pdf	Quick guide to FortranLint	(PDF format)
flintdemo.pdf	Get running quickly with Flint	(PDF format)

The installer copies these files to the following directory:
    C:\Cleanscape\FLINT\Doc

A.1 Installation Procedure, Unix/Linux <u>GUI</u>

Download the file `flintgui<ver>_<OS>.taz` to /tmp. Use the following commands to extract. NOTE: If you are installing to a system directory, you will need to have admin privileges. The '#' below represents the root prompt.

```
# gunzip /tmp/flintgui<ver>_<OS>.taz
# tar xpvf /tmp/flintgui<ver>_<OS>.tar
```

For Linux and Mac, a demo version of Flint will be installed. The demo version requires no key and is thus suitable for evaluation customers. It is fully functional, as can be evidenced by reviewing the summary listing it provides. However, only a limited number of detailed messages, pinpointing the error to the line of code in the source, are provided.

If you have paid for Flint, run the utility ConvertToFull located in your 'main' subdirectory.

To also run on the command line, add the environment variables as detailed in the next section.

To more details, please see section 2.2 of flintguide.pdf, located in your 'doc' subdirectory.

A.2 Installation Procedure, Unix/Linux <u>Command Line</u>

NOTE: You do not need both command line and GUI distros if you want to use both products on your system. The GUI version contains the command line version and only environment variables need to be set.

0.   For command-line-only users, download file `flint<ver>_<OS>.taz` to `/tmp`.

1.   Log in as admin. This is not necessary unless you want to install to a system directory.

2.   Run these commands:
```
# gunzip /tmp/flint<ver>_<OS>.taz
# tar xpvf /tmp/flint<ver>_<OS>.tar
```

3.   The commands will extract a number of files in multiple subdirectories, including:

demo.f	demo source files
demo.inc	demo include file
demo90.f90	demo90 source files
demo90.inc	demo90 include file
flint.1	man page
flint.cfg	flint default configuration file
flint.err	error message text
flint.hls	flint help file
unixlib.lsh	standard UNIX library description text
unixlib.lbt	standard UNIX library file
vmslib.lsh	standard VMS library description text
vmslib.lbt	standard VMS library file

The basic executables include:

flint	FortranLint executable
iptlma	license administration program
iptlmd	license manager daemon
iptlmr	license usage report generator
elmalert	license manager alerts (e.g., key about to expire)

Multiple versions of **flint** may be loaded. For example, **flint_sun4** is the executable for SUN 4 systems. The installation procedure will select the correct version and rename it appropriately.

If the Xlint option was purchased, the following additional files will be loaded:

demo.fdb	database generated from demo.f
demo90.fdb	database generated from demo90.f90
xlint	Xlint executable
Xlint	Xlint resource file

For Linux and Mac, a demo version of Flint will be installed. The demo version requires no key and is thus suitable for evaluation customers. It is fully functional, as can be evidenced by reviewing the summary listing it provides. However, only a limited number of detailed messages, pinpointing the error to the line of code in the source, are provided.

If you have paid for Flint, run the utility ConvertToFull located in your 'main' subdirectory.

4.   Modify the user configuration for each FortranLint user as follows:

(a)   Set the environment variable **FLINTHOST** to the host name for the server where the license manager is installed. (To obtain the host name, execute the UNIX command **hostname** on the server.)

(b)   Set the environment variable **FLINTHOME** to a full path for the directory, which contains the FortranLint support, files (**flint.err**, etc.).

For example, if the user is using **csh**, use commands of the form:

```
setenv FLINTHOST    hostname
setenv FLINTHOME    installation_directory
```

If the user is using **sh**, use commands of the form:

```
FLINTHOST=hostname; export FLINTHOST
FLINTHOME=installation_directory; export FLINTHOME
```

5.   Add **$FLINTHOME** to the user's search list.

For **csh** users, use a command of the form:

        set path=($path $FLINTHOME)

For **sh** users, use a command of the form:

        PATH=$FLINTHOME:$PATH

To make the changes permanent, add the new commands to the appropriate login scripts. For example, for **csh** users, modify "**.cshrc**".

6.   Optional: The FortranLint package includes a utility program named **flpatch** that can be used to patch the FortranLint installation directory and server host name directly into the **flint** executable.

To patch the executable, use commands of the form:

        flpatch flint host   *hostname*
        flpatch flint home   *installation_directory*

For additional information on **FLPATCH**, see section A.4.

7.   Users are now ready to activate FortranLint.

A.3 Activation Procedure, Unix/Linux

Every FortranLint license must be assigned a unique authorization number (or "activation key") by Cleanscape before the package will run.

1.   To proceed, execute the following command:

         flint activate

     The software will provide users with a server code, and it will prompt them to call or email Cleanscape for activation. If users have not already received an activation key, they will need to contact Cleanscape. Cleanscape will then generate an activation key based on the server code.

2.   Once the activation key is acquired, execute the following command:

         flint activate

     again, and enter the activation code when prompted.

     Note: A file called **02** or **07** will be created in the FortranLint installation directory. This file stores information related to the activation key.

3.   The activation procedure in step 2 also creates a script file called **startup** under the installation directory. This file will allow users to load the daemon from the command line. Users will need to run **startup** every time they reboot their system or kill the license manager.

     After the key file has been successfully installed in the installation directory, FortranLint will ask users if they want to put a command to start the license daemon in the system boot file (/etc/rc.local).

     If users answer "yes", the license manager daemon will be started automatically whenever the system is booted. To complete the installation procedure, in this case, simply reboot the system. However, users must have sufficient privileges to do so.

     If users answer "no", they will need to start the daemon manually if they reboot the system or kill the daemon.

4.   If users have elected not to have the boot file modified, they need to run:

         $FLINTHOME/startup

     at the next prompt to start the daemon. After being started, the daemon requires a three-minute period for initialization.

5.   Enter the command:

        flint

FortranLint should display a "help" screen similar to the following:

```
FORTRAN-lint    Rev 7.33                                    07-Nov-2018  12:55:27

Usage:  flint [switches] [file1 [file2...]] [file.lbt] [file.fdb]

 Source configuration options:          Diagnostic options:
  -d       process debug lines           -a       ANSI compatibility
  -e       132 column source lines       -f       report FYIs
  -I path  include file search path      -g       global processing
  -p       preprocess sources (cpp)      -m       flag implicit types
  -R form  source form                   -O num   omit selected messages
  -V sys   select Fortran dialect        -P sys   portability issues
  -2       two byte ints & logicals      -u       data usage checking (dflt)
  -7       set language to FORTRAN 77    -w       report warning      (dflt)
  -9       set language to Fortran 90    -F       Dataflow analysis
  -3       set language to Fortran 2003           (-Fhelp for more info)
  -# file  preprocessor filename (fully qualified if not in PATH)

 Output control options:                Miscellaneous options:
  -l       source listing                -C opts  generate Cadre data files
  -i       show include files            -D defs  #defines for preprocessor
  -s       statistics                    -E file  expand file on cmmd line
  -W num   set page width                -M opts  miscellaneous options
  -Y num   set page length                        (-Mhelp for more info)
  -S file  split output to files         -L file  library generation mode
  -+       summary mode (implies -S)      -q        do not wait for license
  -o 'fmt' specify output format         -B file  create database (.fdb)
  -x       cross reference
  -X opts  cross reference format/content (-Xhelp for more info)
  -t       call tree
  -T opts  call tree format/content       (-Thelp for more info)

Lowercase options may be combined, use double dash to disable (--w).
Uppercase options take parameters (w/ or w/o space) and do not combine.
```

6.   FortranLint is now ready for use.

        Note: At this point, Xlint may be installed. For additional information, see
        Appendix H.

A.4 Patching FortranLint (Unix/Linux only)

Some of the default parameters in FortranLint can be modified within the
FortranLint executable. This is accomplished using the "**flpatch**" program. This
program takes the following command-line arguments:

        1st: name of the executable (required)
        2nd: parameter to patch
        3rd: desired value

If the second or third arguments are not given, the user will be prompted for
them.

The patchable parameters in FortranLint are:

Name	Description	Default value
home	installation directory	/usr/local/flint
host	license server host	?
preprocessor	path of C preprocessor	/usr/lib/cpp
bootfile	system boot file	/etc/rc.local

(a)   The **home** parameter sets the default for the directory of the FortranLint support files.

      home can be overridden by the environment variable **FLINTHOME**.

(b)   The **host** parameter is used by the license manager to locate the machine on which the license manager daemon is running. This helps improve performance since the application does not have to search the entire network for the location of the daemon. This parameter is set during the installation procedure and need not change, except in the event of changing the hostname of the current machine.

      host can be overridden by the environment variable **FLINTHOST**.

      For the daemon hostname, there are two special values:

          (1) a **?** causes a system-wide search for the daemon
          (2) a **!** causes an error to be reported unless **FLINTHOST** is set

(c)   The **preprocessor** parameter allows FortranLint to pass the source code through a preprocessor, usually **cpp**, before analyzing it. The preprocessor must take the same arguments as **cpp**.

(d)   The **bootfile** parameter sets the name of the boot file, which can optionally be modified during the activation procedure to automatically start the license daemon when rebooting the system.

Appendix **B**

❖❖❖
❖❖❖

Installation Under VMS

B.1 Pre-installation

Starting with revision 2.82, FortranLint incorporates a license manager that requires a detached process to be loaded before the product will run. The daemon is called **iptlm** for versions below 2.90, and **iptlmd** for versions 2.90 and above.

Installation can be done by non-privileged users. However, root privileges are required if the product is to be installed in system directories.

Note: The FortranLint "installation directory" mentioned in the following sections is the directory that contains the FortranLint support files (for example, **flint.err** and **flint.cfg)**.

B.2 Installation Procedure

1. Log in as system manager.

2. Create an installation directory. Go to the new directory.

3. Load the tape (or other media) provided and execute commands of the following form:
    ```
    MOUNT /FOR     device_name
    BACKUP /LOG  device_name:FLINT [ ]
    DISMOUNT     device_name
    ```

    where *device_name* is the VMS device name for the media used (TK-50 cartridge tape, 1600 bpi mag tape, etc.).

4. Step 3 will load a number of files, including:

DEMO.FOR	demo source files
DEMO.INC	demo include file
DEMO90.F90	demo90 source files
DEMO90.INC	demo90 include file
FLINT.CFG	FLINT default configuration file
FLINT.ERR	error message text
FLINT.HLP	FLINT help file, use with VMS HELP
FLINT.HLS	FLINT help file, command line options
VMSLIB.LSH	standard VMS library description text
VMSLIB.LBT	standard VMS library file
UNIXLIB.LSH	standard UNIX library description text
UNIXLIB.LBT	standard UNIX library file

The basic executables include:

DEMO.COM	demo script
DEMO90.COM	demo90 script
IPTLMA.EXE	license administration program
IPTLMD.EXE	license manager detached process
IPTLMR.EXE	license usage report generator
FLINT.EXE	FortranLint executable
FLPATCH.EXE	executable patch program

If the Xlint option was purchased, the following additional files will be loaded:

DEMO.FDB	database file for demo.for
DEMO90.FDB	database file for demo90.f90
XLINT.DAT	Xlint resource file
XLINT.EXE	Xlint executable file

5.   Modify the user configuration for each FortranLint user as follows:

(a)   If the FortranLint license manager is installed on a DECNET server, set the logical **FLINTHOST** to the node name for the server. Otherwise, set **FLINTHOST** to "**NO_DECNET**".

Note: To obtain the node name, execute the command "show logical **SYS$NODE**" on the server. Discard any "colon" characters.

define FLINTHOST "*node_name*"

(b)   Set the logical **FLINTHOME** to a full path for the FortranLint installation directory.

define FLINTHOME $*disk*:[*installation_directory*]

(c)   Define the symbol **FLINT** as a foreign command to execute **FLINT.EXE** (located in the installation directory):

FLINT :== $FLINTHOME:FLINT.EXE

(d)   Define the symbol **FLPATCH** as a foreign command to execute **FLPATCH.EXE** (located in the installation directory):

FLPATCH :== $FLINTHOME:FLPATCH.EXE

To make the changes permanent, add the new commands to the appropriate **LOGIN.COM** files.

Example:

```
DEFINE  FLINTHOST  "GUMBY"
DEFINE  FLINTHOME  $disk3:[USR.PETER.FLINT]
FLINT      :==  $FLINTHOME:FLINT.EXE
FLPATCH :==  $FLINTHOME:FLPATCH.EXE
```

6.   Optional: The FortranLint package includes a utility program named **FLPATCH.EXE** that can be used to patch the host directory path and server node name directly into the **FLINT.EXE** executable.

To patch the executable, use commands of the form:

FLPATCH FLINT.EXE HOME $*disk*:[*directory_path*]
FLPATCH FLINT.EXE HOST "*node_name*"

disk:[*directory_path*] should specify the FortranLint installation directory. *node_name* should be the appropriate node name (or "NO_DECNET"), as explained in step 5.

For additional information on **FLPATCH**, see section B.4.

7.   Users are now ready to activate FortranLint.

B.3 Activation Procedure

Each FortranLint license must be assigned a unique authorization number (or "activation key") by Cleanscape before the package will run.

1.   To proceed, execute the following command:

flint /license=activate

The software will provide users with a server code, and it will prompt them to call Cleanscape for activation. If users have not already received an activation key, they will need to contact Cleanscape. Cleanscape will provide an activation key based on the server code.

2.   After the activation code is obtained, execute the command:

flint /license=activate

again, and enter the activation code when prompted.

Note: A file called **02.lic** or **07.lic** will be created in the FortranLint installation directory. This file stores information related to the activation key.

3.   The activation procedure in step 2 also creates a script file called **STARTUP.COM** in the installation directory. This file will allow users to load the daemon from the command file. To start the license daemon, execute the following command:

@FLINTHOME:startup

Users will need to run **@FLINTHOME:STARTUP** if they reboot the system or kill the detached process. Alternatively, add this command to the appropriate system startup script.

4.   The startup script in step 3 requires a three-minute period for initialization. When the three minutes are up, enter the command:

        flint

FortranLint should display a "help" screen:

```
FORTRAN-lint Rev 5.0                           6-Mar-99 10:49:55   Page 1

Source configuration options:                    Diagnostic options:
/DLINES      -- process debug lines            /ANSI        -- ANSI compatibility
/EXTEND      -- 132 column source lines        /FYI         -- report FYIs
/FORM=s      -- source form                    /GLOBAL      -- global processing
/LANG=c      -- select language                /IMPLICIT    -- flag implicit types
/NOI4        -- two byte ints & logicals       /PORT=sys    -- portability issues
/PATH=path   -- include file search path       /NOUSAGE     -- suppress usage checking
/SYS=sys     -- target compiler / system       /NOWARN      -- suppress warnings
/SUPP=n      -- suppress message #n

Output control options:                        Miscellaneous options:
/LIST        -- source listing                 /FILES=f     -- filename/options file
/INCLUDE     -- show include files             /LIB=f       -- library generation mode
/STAT        -- statistics                     /MISC=opt    -- miscellaneous options
/WIDTH=n     -- set page width                 /MISC=help   -- for more info
/LPP=n       -- set page length                /UNIXHELP    -- UNIX-style option help
/OUT=f       -- redirect output to file        -?           -- UNIX-style option help
/SPLIT=f     -- split output to files
/DATA=f      -- create database (.FDB)
/SUMMARY     -- summary mode (implies /SPLIT)
/XREF(=c)    -- cross reference (/XREF=help for more info)
/TREE(=c)    -- call tree (/TREE=help for more info)
```

5.   FortranLint is now ready for use.

      Note: At this point, Xlint may be installed. For additional information, see Appendix I.

B.4 Patching FortranLint

Some of the default parameters in FortranLint can be modified within the FortranLint executable. This is accomplished using the **FLPATCH** program. This program takes three command-line arguments:

        1st: name of the executable (required)
        2nd: parameter to patch
        3rd: desired value

If the second or third arguments are not given, the user will be prompted for them.

The patchable parameters in FortranLint are:

Name	Description	Default value
home	installation directory	"!"
host	license server host	"NO_DECNET"

To patch FortranLint, use a command of the following form:

        FLPATCH flint.exe home $disk2:[appl.FLINT]

a)   The **home** parameter sets the default for the application support directory.

   home can be overridden by the system logical **FLINTHOME**.

   The default value "**!**" for the **home** parameter indicates that the installation directory is not specified. Unless the **FLINTHOME** logical is specified, an error message will be issued reporting that the product is not yet installed.

b)   The **host** parameter is used by the license manager to locate the machine on which the license manager detached process is running.

   If the license manager is not installed on a DECNET server, this parameter should be set to "**NO_DECNET**".

   The **host** parameter can be overridden by setting the logical **FLINTHOST**.

   For the detached process hostname, there are two special values:

   1) a **?** causes a system-wide search for the detached process.
   2) a **!** causes an error to be reported unless **FLINTHOST** is set.

Appendix C

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

License Manager, Unix/Linux/VMS

Starting with version 2.81, FortranLint incorporates a license manager that requires a daemon to be loaded before the product will run. This daemon is called **iptlm** for versions below 2.90 and **iptlmd** for versions 2.90 and above.

Most of the information here is relevant for non-Windows users.

C.1 License Manager Commands

C.1.1 User Commands

1) By default, FortranLint will queue a job when there are no more licenses available. The "**-q**" option or "**/quit**" under VMS when added to the "**flint**" command line will force the application to quit when there are no available licenses.

>   flint -q [*options*] files
> or
>   flint /quit [*options*] files   under VMS

2) FortranLint needs to identify the node on which the license manager was loaded. To accomplish this, it uses the UNIX environment variable (or VMS logical) **FLINTHOST**.

To define **FLINTHOST**, use commands of the form:

>   setenv FLINTHOST hostname   for **csh** users
> or
>   FLINTHOST=*hostname*   for **sh** users
>   export FLINTHOST

hostname should be the network name of the user's license server host. (To obtain the host name , execute the command **hostname** on the server.)

 To define **FLINTHOST** under VMS, use a command of the form:

>   define FLINTHOST "*nodename*"

If the license server is installed on a DECNET host, *nodename* should be the node name of the host. Otherwise, *nodename* should be "**NO_DECNET**".

(To obtain the node name under VMS, execute the command "show logical **SYS$NODE**" on the server. Discard any "colon" characters.)

3)   Another environment variable (or logical) **FLINTHOME** tells FortranLint where the installation directory is located. This variable can be used to override the directory value patched into the executable during installation.

To define **FLINTHOME**, use commands of the form:

>   setenv FLINTHOME *directory*   for **csh** users
> or
>   FLINTHOME=*directory*   for **sh** users
>   export FLINTHOME

directory should be a full path.

To define **FLINTHOME** under VMS, use a command of the form:

>   define FLINTHOME $*disk*:[*installation_directory*]

4)   Under VMS, the qualifier **/system** may be added to the commands in 2) and 3) to place the definitions of **FLINTHOST** and **FLINTHOME** in the system logical table. Note that users need to log in with SYSNAM privileges to add definitions to the system logical table.

C.1.2 Administrative Commands

	Under VMS	Description
flint activate	flint /license=activate	Enter an activation key
flint users	flint /license=users	Show active users (outstanding licenses)
flint servers	not supported	List active license daemons
flint report	flint /license=report	Produce cumulative usage report
flint daily	flint /license=daily	Produce daily usage report
flint kill	flint /license=kill	Kill license daemon (superuser)

C.1.3 License Manager Options (at daemon startup only)

1) **iptlmd -e dir:dir:dir...**   Key file directories (required)

Directories must be full pathnames separated by colons.

2) **iptlmd -r file**   Reserve file

This file allows licenses to be reserved for specific users or machines.

The format of this file is:

        product:group:client1, ...,clientn:*K*

Each group, with the client members, has *K* licenses for FortranLint.

If users are using UNIX based systems, the group name is unrelated to the UNIX system group names. It is only the name you wish to call this group of users.

A client is either a user name or a host name preceded by the at sign ("@").

Example:

        flint:hackers:wendy,jeff,sara,fred:3
        flint:lab:@gumby:1

Here Fred is a user name, and @gumby is a host name.

Comment lines begin with "#".

3) **iptlmd -l logfile**   Log file (needed for usage reports; recommended).

4)   **iptlmd -m size**   Maximum log file size. This limits the size of the log file. When this size is reached, the log file is copied to *file*.old and is cleared. The size is given as a floating number followed by either "**m**" for megabytes or "**k**" for kilobytes.

Examples: -m 100k
                                          -m 0.5m

5)   **iptlmd -v #**   Log file verbosity. The default value is 3. Lower numbers produce less output and higher numbers produce more output.

6) **iptlmd -f**   Run in foreground. The license manager normally "backgrounds" itself and exits. This option keeps it in the foreground.

C.1.4 elmalert

elmalert allows system administrators or users to receive alerts as to license status, either via email or onscreen. This is particularly useful for admins to be notified of the license key's expiration date. It can also notify regular users when a license becomes available.

Included is a manpage which explains its usage and can be put in a suitable 'man' directory on your license server. Note there are 32- and 64-bit versions of elmalert.

- To see the current status onscreen, type elmalert64 -v 07
- More advanced features include -x (generate message x days before expiration) and -m (email default user) as shown in the last example in the manpage.

Appendix D

❖❖
❖❖

Sample Output: Fortran 90

D.1 Sample Fortran 90 Program

```fortran
MODULE M
  TYPE MYTYPE
    CHARACTER*10    NAME
    INTEGER(KIND=4) SCORES(2,2)
  END TYPE

  REAL, PRIVATE :: LOC(10)
  INTEGER AVE
  CHARACTER*2 GRADE(6)
  PARAMETER (GRADE = (/'A', 'B', 'C', 'D', 'E', 'F'/) )

CONTAINS
! ------ Internal subprograms --------
  SUBROUTINE M_INNER(TYPE1, TYPE2)
    TYPE (MYTYPE), INTENT(INOUT) :: TYPE1
    TYPE (MYTYPE), INTENT(IN)    :: TYPE2
    TYPE1%NAME = 'ALIAS: ' // TYPE2%NAME
  END SUBROUTINE M_INNER

END MODULE

SUBROUTINE OUTER(TYPE1, TYPE2, OPDUM)
C     ----- Declaration -----
      USE M, ONLY : MYTYPE
      TYPE (MYTYPE), INTENT(INOUT) :: TYPE1, TYPE2
      INTEGER, OPTIONAL :: OPDUM
C     ----- Double TYPE2's scores.
      if ( PRESENT(OPDUM) ) THEN
         TYPE1%SCORES(1, 1) = TYPE2%SCORES(1) * OPDUM
      ELSE
         TYPE1%SCORES(1, 1) = TYPE2%SCORES(1, 1) * 2
      ENDIF
      END SUBROUTINE

PROGRAM MAIN
! --------- Main program --------

  USE M, TYPE_S => MYTYPE, &
       MYLOC  => LOC     !private module entities cannot be accessed
  CHARACTER(LEN = 10) STR
  TYPE (TYPE_S) STUDENT1, STUDENT2

  CALL M_INNER(STUDENT1, STUDENT2)
  CALL OUTER(STUDENT1, STUDENT2)
  STR = GRADE(3)
  AVE = MAIN_INNER( STUDENT1%SCORES )

CONTAINS
! Internal subprograms are in another file
  INCLUDE 'demo90.inc'
END
```

```
! ** demo90.inc **
FUNCTION MAIN_INNER(DUM)
  REAL, INTENT(INOUT) :: DUM(:, :)
  REAL (KIND=KIND(0.0D0)) :: SUM = 0
  DONAME1: DO 10 I = 10, SIZE(DUM, 1)
  DONAME2:   DO 20 J = 10, SIZE(DUM, 2)
    IF (SUM < 0) CYCLE DONAME2
    SUM = SUM + DUM(I, J)
    IF (SUM > 100) EXIT DONAME1
20  END DO DONAME2
10  END DO DONAME1
  MAIN_INNER = INT (SUM)
END FUNCTION
```

D.2 Analysis Output

```
FortranLint   Rev 5.0                          2-Jan-02  10:49:55   Page 1

Default options:  -w -u -O207,276,76,261 -Ttrim -Xno_unreferenced_parameters
                  -Xno_unused_common_variables
Command options:  -f -g -s -t -x -Sdemo90

demo90.f90

*******************************************************************************
   Subroutine M_INNER                   File demo90.f90          Line 16
           <Module subprog of M>

>    TYPE1%NAME = 'ALIAS: ' // TYPE2%NAME
>            ^
demo90.f90:M_INNER line 19:
SYNTAX FYI #105- string will be truncated (from 17 to 10 chars).

*******************************************************************************
   Subroutine OUTER                     File demo90.f90          Line 25

>         TYPE1%SCORES(1, 1) = TYPE2%SCORES(1) * OPDUM
>                                       ^
demo90.f90:OUTER line 32:
SYNTAX ERROR #168- array referenced with too few subscripts.

*******************************************************************************
      Program MAIN                      File demo90.f90          Line 38

>  USE M, TYPE_S => MYTYPE, &
>       MYLOC  => LOC     !private module entities cannot be accessed
>              ^
demo90.f90:MAIN line 42:
SYNTAX ERROR #661- entity not accessible in module M.

>  CALL M_INNER(STUDENT1, STUDENT2)
>           ^
demo90.f90:MAIN line 46:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>  CALL M_INNER(STUDENT1, STUDENT2)
>                    ^
demo90.f90:MAIN line 46:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>  CALL OUTER(STUDENT1, STUDENT2)
>            ^
```

```
demo90.f90:MAIN line 47:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   CALL OUTER(STUDENT1, STUDENT2)
>                       ^
demo90.f90:MAIN line 47:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   AVE = MAIN_INNER( STUDENT1%SCORES )
>                     ^
demo90.f90:MAIN line 49:
INTERFACE ERROR #252- I*4 array passed to dummy arg which is a R*4 array.

demo90.f90:MAIN line 46:
USAGE ERROR #126- local variable STUDENT2 is referenced but never set.

demo90.f90:MAIN line 48:
USAGE WARNING #127- local variable STR is set but never referenced.

*******************************************************************************
    Function MAIN_INNER                  File demo90.f90           Line 53
            <Internal subprog of MAIN>

*******************************************************************************

Global checking:

USAGE WARNING #743- module entity set but not referenced:  M:AVE

USAGE FYI #744- unused module entity:  M:LOC
```

Under VMS:

```
FortranLint   Rev 4.30                           2-Jan-02  10:49:55   Page 1

Local options:  /WARNINGS /USAGE /SUPPRESS=207,276,76,261 /NOTREE /NOXREF
Command options:  /FYI /GLOBAL /STATISTICS /OUTPUT=demo90

DEMO90.F90;403

*******************************************************************************
   Subroutine M_INNER                  File DEMO90.F90           Line 16
            <Module subprog of M>

>    TYPE1%NAME = 'ALIAS: ' // TYPE2%NAME
>               ^
DEMO90.F90:M_INNER line 19:
SYNTAX FYI #105- string will be truncated (from 17 to 10 chars).

*******************************************************************************
   Subroutine OUTER                    File DEMO90.F90           Line 25

>         TYPE1%SCORES(1, 1) = TYPE2%SCORES(1) * OPDUM
>                                              ^
DEMO90.F90:OUTER line 32:
SYNTAX ERROR #168- array referenced with too few subscripts.

*******************************************************************************
     Program MAIN                      File DEMO90.F90           Line 38

> USE M, TYPE_S => MYTYPE, &
>       MYLOC  => LOC     !private module entities cannot be accessed
>                  ^
```

```
DEMO90.F90:MAIN line 42:
SYNTAX ERROR #661- entity not accessible in module M.

>   CALL M_INNER(STUDENT1, STUDENT2)
>              ^
DEMO90.F90:MAIN line 46:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   CALL M_INNER(STUDENT1, STUDENT2)
>                        ^
DEMO90.F90:MAIN line 46:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   CALL OUTER(STUDENT1, STUDENT2)
>            ^
DEMO90.F90:MAIN line 47:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   CALL OUTER(STUDENT1, STUDENT2)
>                      ^
DEMO90.F90:MAIN line 47:
INTERFACE FYI #256- type TYPE_S passed to a type MYTYPE dummy arg (same format
but different names).

>   AVE = MAIN_INNER( STUDENT1%SCORES )
>                   ^
DEMO90.F90:MAIN line 49:
INTERFACE ERROR #252- I*4 array passed to dummy arg which is a R*4 array.

DEMO90.F90:MAIN line 46:
USAGE ERROR #126- local variable STUDENT2 is referenced but never set.

DEMO90.F90:MAIN line 48:
USAGE WARNING #127- local variable STR is set but never referenced.

****************************************************************************
     Function MAIN_INNER                 File DEMO90.F90           Line 53
              <Internal subprog of MAIN>

****************************************************************************

Global checking:

USAGE WARNING #743- module entity set but not referenced:  M:AVE

USAGE FYI #744- unused module entity:  M:LOC
```

D.3 Statistics Output

```
>>> Statistics:

 Number of source files:     1

 Source files:        54 lines,      1273 bytes   ( 18% comments, 82% code )
 Include files:       14 lines,       352 bytes   (  5% comments, 95% code )
 Total parsed:        68 lines,      1625 bytes   ( 15% comments, 85% code )

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

Individual message summary
--------------------------
INTRFC FYI #256-    4x: * passed to a * dummy arg (same format but different
                        names).
SYNTAX FYI #105-    1x: string will be truncated (from * to * chars).
USAGE ERR #126-     1x: local variable * 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 #252-    1x: * array passed to dummy arg which is a * array.
SYNTAX ERR #661-    1x: entity not accessible in module *.
USAGE WARN #743-    1x: module entity set but not referenced:  *, *
USAGE FYI #744-     1x: unused module entity:  *, *

Total messages: 12

                  Errors   Warnings   FYIs
                 -------- -------- --------
    Syntax:          2        0        1
    Interface:       1        0        4
    Data usage:      1        2        1

    Implicit typing:     <supp>
```

D.4 Call Tree

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

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

D.5 Freeform Cross Reference

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

*** Program:

MAIN : defined at line 38 of demo90.f90
                    Calls- demo90.f90:M, demo90.f90:M::M_INNER,
                           demo90.f90:OUTER, demo90.f90:MAIN::MAIN_INNER


*** Subroutines:

M_INNER : M internal : defined at line 16 of demo90.f90
                    Args- (type MYTYPE S, type MYTYPE R)
                    Called by- demo90.f90:MAIN
OUTER : defined at line 25 of demo90.f90
                    Args- (type MYTYPE S, type MYTYPE R, I*4 RO)
                    Calls- demo90.f90:M
                    Called by- demo90.f90:MAIN


*** Functions:

INT : I*4 : intrinsic function
                    Called by- demo90.f90:MAIN::MAIN_INNER
KIND : I*4 : intrinsic function
                    Called by- demo90.f90:MAIN::MAIN_INNER
MAIN_INNER : I*4 : MAIN internal : defined at line 53 of demo90.f90
                    Args- (R*4 array R)
                    Called by- demo90.f90:MAIN
PRESENT : L*4 : intrinsic function
                    Called by- demo90.f90:OUTER
SIZE : I*4 : intrinsic function
                    Called by- demo90.f90:MAIN::MAIN_INNER


*** Modules:

M : defined at line 3 of demo90.f90
                    Called by- demo90.f90:OUTER, demo90.f90:MAIN


*** Types:

MYTYPE : size = 26 bytes
     NAME : CHAR*10
          in (demo90.f90:M) is Unused
          in (demo90.f90:M::M_INNER) is Ref, Set
          in (demo90.f90:OUTER) is Unused
     SCORES (2,2) : I*4 : KIND= 4
          in (demo90.f90:M) is Unused
          in (demo90.f90:OUTER) is Ref, Set
TYPE_S : size = 26 bytes
     NAME : CHAR*10
          in (demo90.f90:MAIN) is Unused
     SCORES (2,2) : I*4 : KIND= 4
          in (demo90.f90:MAIN) is Ref, Actual arg


*** Records:

STUDENT1 : type TYPE_S : local
          in (demo90.f90:MAIN) is Ref, Set, Actual arg
STUDENT2 : type TYPE_S : local
          in (demo90.f90:MAIN) is Ref, Actual arg
TYPE1 : type MYTYPE : local
          in (demo90.f90:M::M_INNER) is Dummy arg, Set
          in (demo90.f90:OUTER) is Dummy arg, Set
TYPE2 : type MYTYPE : local
          in (demo90.f90:M::M_INNER) is Dummy arg, Ref
          in (demo90.f90:OUTER) is Dummy arg, Ref
```

```
*** Vars/Arrays:

AVE : I*4 : public entity of module M
          in (demo90.f90:M) is Unused
          in (demo90.f90:MAIN) is Set
DUM (:,:) : R*4 : local
          in (demo90.f90:MAIN::MAIN_INNER) is Dummy arg, Ref
I : I*4 : local
          in (demo90.f90:MAIN::MAIN_INNER) is Ref, Set
J : I*4 : local
          in (demo90.f90:MAIN::MAIN_INNER) is Ref, Set
LOC (10) : R*4 : private entity of module M
          in (demo90.f90:M) is Unused
OPDUM : I*4 : local
          in (demo90.f90:OUTER) is Dummy arg, Ref
STR : CHAR*10 : local
          in (demo90.f90:MAIN) is Set
SUM : R*8 : KIND= 8 : local
          in (demo90.f90:MAIN::MAIN_INNER) is Ref, Set, Initialized

*** Parameters:

GRADE (6) : CHAR*2
          in (demo90.f90:MAIN) is Ref
```

D.6 Tabular Cross Reference

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

*** Functions:
```

| | | | | | /---Calls---\ | /----------References----------\ | | |
Name	Class	Type	Definition	Arguments	Line-Subprog	Subprog	File	Line
INT	intrinsic func	I*4				MAIN_INNER	demo90.f90	13
KIND	intrinsic func	I*4				MAIN_INNER	demo90.f90	5
M	module		demo90.f90 line 3			OUTER	demo90.f90	27
						MAIN	demo90.f90	41
MAIN	program		demo90.f90 line 38		41-M			
					46-M_INNER			
					47-OUTER			
					49-MAIN_INNER			
MAIN_INNER	function	I*4	demo90.f90 line 53	1:(R*4 array R)		MAIN	demo90.f90	49
	MAIN internal							
M_INNER	subroutine		demo90.f90 line 16	1:(type MYTYPE S)		MAIN	demo90.f90	46
	M internal			2:(type MYTYPE R)				
OUTER	subroutine		demo90.f90 line 25	1:(type MYTYPE S)	27-M	MAIN	demo90.f90	47
				2:(type MYTYPE R)				
				3:(I*4 RO)				
PRESENT	intrinsic func	L*4				OUTER	demo90.f90	31
SIZE	intrinsic func	I*4				MAIN_INNER	demo90.f90	6
						MAIN_INNER	demo90.f90	7

```
*** Types:
```

| | | /------------------------------------Fields--\ | | | | | | |
Name	Size	Field	Type	Kind	Attributes	Subprogram	File	References	
MYTYPE	26	NAME	CHAR*10			M	demo90.f90	5-D	
						M::M_INNER	demo90.f90	19-S	19-R
		SCORES	I*4 (2,2)	4		M	demo90.f90	6-D	

```
                                            OUTER           demo90.f90      32-S      32-R      34-S
                                                                            34-R
----------------------------------------------------------------------------------------------------------
TYPE_S      26      NAME        CHAR*10
----------------------------------------------------------------------------------------------------------
                    SCORES      I*4 (2,2)           4           MAIN            demo90.f90      49-RA
----------------------------------------------------------------------------------------------------------


*** Vars/Arrays:

Name        Type                Kind  Attributes          Subprogram      File          References
----------------------------------------------------------------------------------------------------------
AVE         I*4                       public entity of    M               demo90.f90      10-D
                                      module M            MAIN            demo90.f90      49-S
----------------------------------------------------------------------------------------------------------
DUM         R*4 (:,:)                 local               MAIN::MAIN_INNER
                                                          demo90.f90   (demo90.inc)3-P  (demo90.inc)4-D
                                                             (demo90.inc)6-R  (demo90.inc)7-R
                                                             (demo90.inc)9-R
----------------------------------------------------------------------------------------------------------
I           I*4                       local               MAIN::MAIN_INNER
                                                          demo90.f90   (demo90.inc)6-RS (demo90.inc)9-R
----------------------------------------------------------------------------------------------------------
J           I*4                       local               MAIN::MAIN_INNER
                                                          demo90.f90   (demo90.inc)7-RS (demo90.inc)9-R
----------------------------------------------------------------------------------------------------------
LOC         R*4 (10)                  private entity of   M               demo90.f90       9-D
                                      module M
----------------------------------------------------------------------------------------------------------
OPDUM       I*4                       local               OUTER           demo90.f90      25-P      29-D      31-R      32-R
----------------------------------------------------------------------------------------------------------
STR         CHAR*10                   local               MAIN            demo90.f90      43-D      48-S
----------------------------------------------------------------------------------------------------------
STUDENT1    type TYPE_S               local               MAIN            demo90.f90      44-D      46-SA     47-SA     49-RA
----------------------------------------------------------------------------------------------------------
STUDENT2    type TYPE_S               local               MAIN            demo90.f90      44-D      46-RA     47-RA
----------------------------------------------------------------------------------------------------------
SUM         R*8                 8     local               MAIN::MAIN_INNER
                                                          demo90.f90   (demo90.inc)5-D  (demo90.inc)5-I
                                                             (demo90.inc)8-R  (demo90.inc)9-S
                                                             (demo90.inc)9-R (demo90.inc)10-R
                                                             (demo90.inc)13-R
----------------------------------------------------------------------------------------------------------
TYPE1       type MYTYPE               local               M::M_INNER      demo90.f90      16-P      17-D      19-S
                                                          OUTER           demo90.f90      25-P      28-D      32-S      34-S
----------------------------------------------------------------------------------------------------------
TYPE2       type MYTYPE               local               M::M_INNER      demo90.f90      16-P      18-D      19-R
                                                          OUTER           demo90.f90      25-P      28-D      32-R      34-R
```

```
--------------------------------------------------------------------------------------------------------------

*** Parameters:

Name            Type                   Kind  Value               Subprogram   File          References
--------------------------------------------------------------------------------------------------------------
GRADE           CHAR*2 (6)                                       MAIN         demo90.f90    48-R
--------------------------------------------------------------------------------------------------------------


                            ----------------- LEGEND ------------------

                              A - actual argument
                              B - used as an assumed array bound
                              C - Associated
                              c - Loop Counter
                              D - declaration
                              E - equivalenced
                              F - statement function dummy argument
                              G - used as a label in a goto statement
                              I - initialized
                              i - initialized indirectly
                              L - used as a label in an assign statement
                              M - allocated
                              N - nullified
                              O - optional dummy argument
                              P - dummy argument
                              R - referenced
                              S - set
                              X - usage cannot be determined
                              Z - deallocated

                            -------------------------------------------
```

Appendix E

❖❖
❖❖

Sample Output: FORTRAN 77

E.1 Sample FORTRAN 77 Program

```
C  'PROCDAT'
       PROGRAM PROCDAT
       INTEGER IUNIT, PUNIT
       INCLUDE 'demo.inc'
       DO 100 I = 1, 5
 50      CALL GETUNIT( I+5, IUNIT, PUNIT)
        CALL READNAME( CURITEM.NAME, CURITEM.DIMENSIONS)
        CALL SETTYPE( CURITEM)
        CALL PRINT( CURITEM, IUNIT)
100     CONTINUE
       IF (IUNIT .EQ. 23) GO TO 50
       END
C 'GETUNIT'
       SUBROUTINE GETUNIT( UNIT, UNIT1)
       INTEGER UNIT, UNIT1
       READ (UNIT1,*) UNIT
       END
C 'READNAME'
       SUBROUTINE READNAME( NAME, DIMS)
       CHARACTER*(*) NAME
       INTEGER INUSE, STATUS
       COMMON /BLOCK/ INUSE, STATUS
       REAL*8 DIMS(3)
       READ (5, *) NAME, DIMS
       END
C 'SETTYPE'
       SUBROUTINE SETTYPE( CURITEM)
       INCLUDE 'demo.inc'
       CURITEM.TYPE = CURITEM.DIMENSIONS(2)
       IF (CURITEM.TYPE .GT. 5) CALL PRINT( CURITEM)
       END
C 'PRINT'
       SUBROUTINE PRINT( CURITEM, IUNIT)
       INCLUDE 'demo.inc'
       IF (CURITEM.TYPE .NE. COUNT) CALL PRINTIT( IUNIT, CURITEM)
       END
C 'PRINTIT'
       SUBROUTINE PRINTIT( IUNIT, CURITEM)
       INCLUDE 'demo.inc'
       IF (IUNIT .EQ. INUSE) THEN
        STATUS = 2
        CALL DIPSTAT( 4, CURITEM)
        CALL GETUNIT( INUIT, 3)
       END IF
       WRITE (IUNIT,*) CURITEM.TYPE
       END
 C 'DIPSTAT'
       SUBROUTINE DIPSTAT( ISTAT, CURITEM)
       ISTAT = PRINT( CURITEM, 1)
       END

C                  <<< DEMO.INC >>>
```

```
        STRUCTURE /ITEM/
         CHARACTER*10 NAME
         INTEGER TYPE
         REAL DIMENSIONS(3)
        END STRUCTURE
        RECORD /ITEM/ CURITEM

        INTEGER INUSE*2, STATUS, COUNT, TIME
        COMMON /BLOCK/ INUSE, STATUS
        COMMON /BK2/ COUNT, TIME
```

E.2 Analysis Output

```
FortranLint   Rev 5.0                          2-Jan-02  10:49:55   Page 1
Default options:  -w -u -O207,276,76,261 -Ttrim -Xno_unreferenced_parameters
                  -Xno_unused_common_variables
Command options:  --f -g -s -t -x -Sdemo -7

demo.f

******************************************************************************
      Program PROCDAT                    File demo.f                Line 2

> 50      CALL GETUNIT( I+5, IUNIT, PUNIT)
>                       ^
demo.f:PROCDAT line 6:
INTERFACE WARNING #63- expression is changed by subprogram.

> 50      CALL GETUNIT( I+5, IUNIT, PUNIT)
>                                   ^
demo.f:PROCDAT line 6:
INTERFACE ERROR #57- too many arguments.

>         CALL READNAME( CURITEM.NAME, CURITEM.DIMENSIONS)
>                                      ^
demo.f:PROCDAT line 7:
INTERFACE ERROR #252- R*4 array passed to dummy arg which is a R*8 array.

>         CALL READNAME( CURITEM.NAME, CURITEM.DIMENSIONS)
>                                      ^
demo.f:PROCDAT line 7:
INTERFACE ERROR #287- R*4 array passed to R*8 array of larger size (by 12
bytes).

>         IF (IUNIT .EQ. 23) GO TO 50
>                                    ^
demo.f:PROCDAT line 11:
SYNTAX WARNING #47- branch into do loop via label 50.

demo.f:PROCDAT line 6:
USAGE ERROR #126- local variable IUNIT is referenced but never set.

demo.f:PROCDAT line 3:
USAGE FYI #128- local variable PUNIT declared but unused.

******************************************************************************
    Subroutine READNAME                  File demo.f                Line 19

demo.f:READNAME line 22:
INTERFACE WARNING #185- common block /BLOCK/ length mismatch (compared to
initial use in routine PROCDAT).
```

```
demo.f:READNAME line 22:
INTERFACE WARNING #122- common block /BLOCK/ organization differs at member
                        INUSE (compared to initial use in routine PROCDAT).

********************************************************************************
   Subroutine SETTYPE                    File demo.f              Line 27

>       IF (CURITEM.TYPE .GT. 5) CALL PRINT( CURITEM)
>                                                    ^
demo.f:SETTYPE line 30:
INTERFACE ERROR #56- not enough arguments.

********************************************************************************
   Subroutine PRINTIT                    File demo.f              Line 38

>          CALL DIPSTAT( 4, CURITEM)
>                        ^
demo.f:PRINTIT line 42:
INTERFACE ERROR #59- constant is changed by subprogram.

>          CALL DIPSTAT( 4, CURITEM)
>                           ^
demo.f:PRINTIT line 42:
INTERFACE ERROR #248- struct ITEM passed to a R*4 dummy arg.

demo.f:PRINTIT line 43:
USAGE WARNING #127- local variable INUIT is set but never referenced.

********************************************************************************
   Subroutine DIPSTAT                    File demo.f              Line 48
>       ISTAT = PRINT( CURITEM, 1)
>               ^
demo.f:DIPSTAT line 49:
INTERFACE ERROR #95- this name is defined as a subroutine.

********************************************************************************
Global checking:

*** Inconsistent organization of common /BLOCK/, ref/set checking suppressed
for this common block

USAGE ERROR #133- common block members referenced but not set:  /BK2/COUNT
USAGE FYI #135- unused common block members:   /BK2/TIME
```

Under VMS:

```
FortranLint   Rev 4.30                              2-Jan-02  10:49:55   Page 1

Local options:  /WARNINGS /USAGE /SUPPRESS=207,276,76,261 /NOTREE /NOXREF
Command options:  /FYI /GLOBAL /STATISTICS /OUTPUT=demo /LANG=f77

DEMO.F;403

********************************************************************************
    Program PROCDAT                      File DEMO.F              Line 2
> 50     CALL GETUNIT( I+5, IUNIT, PUNIT)
>                      ^
DEMO.F:PROCDAT line 6:
INTERFACE WARNING #63- expression is changed by subprogram.

> 50     CALL GETUNIT( I+5, IUNIT, PUNIT)
>                                  ^
DEMO.F:PROCDAT line 6:
INTERFACE ERROR #57- too many arguments.
```

```
>          CALL READNAME( CURITEM.NAME, CURITEM.DIMENSIONS)
>                                         ^
DEMO.F:PROCDAT line 7:
INTERFACE ERROR #252- R*4 array passed to dummy arg which is a R*8 array.

>          CALL READNAME( CURITEM.NAME, CURITEM.DIMENSIONS)
>                                         ^
DEMO.F:PROCDAT line 7:
INTERFACE ERROR #287- R*4 array passed to R*8 array of larger size (by 12
bytes).

>       IF (IUNIT .EQ. 23) GO TO 50
>                          ^
DEMO.F:PROCDAT line 11:
SYNTAX WARNING #47- branch into do loop via label 50.

DEMO.F:PROCDAT line 6:
USAGE ERROR #126- local variable IUNIT is referenced but never set.

DEMO.F:PROCDAT line 3:
USAGE FYI #128- local variable PUNIT declared but unused.

*****************************************************************************
    Subroutine READNAME                 File DEMO.F                 Line 19

DEMO.F:READNAME line 22:
INTERFACE WARNING #185- common block /BLOCK/ length mismatch (compared to
initial
                        use in routine PROCDAT).

DEMO.F:READNAME line 22:
INTERFACE WARNING #122- common block /BLOCK/ organization differs at member
INUSE
                        (compared to initial use in routine PROCDAT).

*****************************************************************************
    Subroutine SETTYPE                  File DEMO.F                 Line 27

>       IF (CURITEM.TYPE .GT. 5) CALL PRINT( CURITEM)
>                                                ^
DEMO.F:SETTYPE line 30:
INTERFACE ERROR #56- not enough arguments.

*****************************************************************************
    Subroutine PRINTIT                  File DEMO.F                 Line 38

>          CALL DIPSTAT( 4, CURITEM)
>                        ^
DEMO.F:PRINTIT line 42:
INTERFACE ERROR #59- constant is changed by subprogram.

>          CALL DIPSTAT( 4, CURITEM)
>                           ^
DEMO.F:PRINTIT line 42:
INTERFACE ERROR #248- struct ITEM passed to a R*4 dummy arg.

DEMO.F:PRINTIT line 43:
USAGE WARNING #127- local variable INUIT is set but never referenced.

*****************************************************************************
    Subroutine DIPSTAT                  File DEMO.F                 Line 48

>       ISTAT = PRINT( CURITEM, 1)
>               ^
DEMO.F:DIPSTAT line 49:
INTERFACE ERROR #95- this name is defined as a subroutine.
```

```
********************************************************************************
Global checking:

*** Inconsistent organization of common /BLOCK/, ref/set checking suppressed
for this common block

USAGE ERROR #133- common block members referenced but not set:  /BK2/COUNT
USAGE FYI #135- unused common block members:  /BK2/TIME
```

E.3 Statistics Output

```
>>> Statistics:

 Number of source files:      1

 Source files:        50 lines,      1276 bytes   (  6% comments, 94% code )
 Include files:       44 lines,      1052 bytes   ( 14% comments, 86% code )
 Total parsed:        94 lines,      2328 bytes   ( 10% comments, 90% code )

 Total subprograms:      7
   Subroutines:          6
   Functions:            0
   Program:              1
   Block Data:           0
   Module:               0

Individual message summary
--------------------------
SYNTAX WARN #47-    1x: branch into do loop via label *.
INTRFC ERR #56-     1x: not enough arguments.
INTRFC ERR #57-     1x: too many arguments.
INTRFC ERR #59-     1x: constant is changed by subprogram.
INTRFC WARN #63-    1x: expression is changed by subprogram.
INTRFC ERR #95-     1x: this name is defined as a subroutine.
INTRFC WARN #122-   1x: common block /*/ organization differs at member *
                        (compared to initial use in routine *).
USAGE ERR #126-     1x: local variable * is referenced but never set.
USAGE WARN #127-    1x: local variable * is set but never referenced.
USAGE FYI #128-     1x: local variable * declared but unused.
USAGE ERR #133-     1x: common block members referenced but not set:  *, *
USAGE FYI #135-     1x: unused common block members:  *, *
INTRFC WARN #185-   1x: common block /*/ length mismatch (compared to initial
                        use in routine *).
INTRFC ERR #248-    1x: * passed to a * dummy arg.
INTRFC ERR #252-    1x: * array passed to dummy arg which is a * array.
INTRFC ERR #287-    1x: * array passed to * array of larger size (by * bytes).

Total messages: 16

                    Errors   Warnings   FYIs
                   -------- -------- --------
     Syntax:           0        1        0
     Interface:        7        3        0
     Data usage:       2        1        2

     Implicit typing:     <supp>
```

E.4 Call Tree

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

```
PROCDAT-+-GETUNIT
        |
        +-READNAME
        |
        +-SETTYPE--PRINT (1)--PRINTIT-+-DIPSTAT--*PRINT*
        |                             |
        |                             +-GETUNIT
        |
        +-PRINT see 1
```

E.5 Freeform Cross Reference

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

*** Program:

PROCDAT : defined at line 2 of demo.f
                   Calls- demo.f:GETUNIT, demo.f:READNAME, demo.f:SETTYPE,
                           demo.f:PRINT
 *** Subroutines:

DIPSTAT : defined at line 48 of demo.f
                   Args- (I*4 S, R*4 R)
                   Calls- demo.f:PRINT
                   Called by- demo.f:PRINTIT
GETUNIT : defined at line 14 of demo.f
                   Args- (I*4 S, I*4 R)
                   Called by- demo.f:PROCDAT, demo.f:PRINTIT
PRINT : defined at line 33 of demo.f
                   Args- (struct ITEM R, I*4 R)
                   Calls- demo.f:PRINTIT
                   Called by- demo.f:PROCDAT, demo.f:SETTYPE, demo.f:DIPSTAT
PRINTIT : defined at line 38 of demo.f
                   Args- (I*4 R, struct ITEM R)
                   Calls- demo.f:DIPSTAT, demo.f:GETUNIT
                   Called by- demo.f:PRINT
READNAME : defined at line 19 of demo.f
                   Args- (CHAR*(*) S, R*8 array S)
                   Called by- demo.f:PROCDAT
SETTYPE : defined at line 27 of demo.f
                   Args- (struct ITEM RS)
                   Calls- demo.f:PRINT
                   Called by- demo.f:PROCDAT


*** Common blocks:

BK2 : size = 8 bytes : Members- COUNT, TIME
                   Defined in- demo.f:PROCDAT, demo.f:SETTYPE, demo.f:PRINT,
                           demo.f:PRINTIT
BLOCK : size = 6 bytes : Members- INUSE, STATUS
                   Defined in- demo.f:PROCDAT, demo.f:READNAME,
                           demo.f:SETTYPE, demo.f:PRINT, demo.f:PRINTIT
```

```
*** Structures:

ITEM : size = 26 bytes
     NAME : CHAR*10
          in (demo.f:PROCDAT) is Set, Actual arg
          in (demo.f:SETTYPE) is Unused
          in (demo.f:PRINT) is Unused
          in (demo.f:PRINTIT) is Unused
     TYPE : I*4
          in (demo.f:PROCDAT) is Unused
          in (demo.f:SETTYPE) is Ref, Set
          in (demo.f:PRINT) is Ref
          in (demo.f:PRINTIT) is Ref
     DIMENSIONS (3) : R*4
          in (demo.f:PROCDAT) is Set, Actual arg
          in (demo.f:SETTYPE) is Ref
          in (demo.f:PRINT) is Unused
          in (demo.f:PRINTIT) is Unused

*** Records:

CURITEM : struct ITEM : local
          in (demo.f:PROCDAT) is Ref, Set, Actual arg
          in (demo.f:SETTYPE) is Dummy arg, Ref, Set, Actual arg
          in (demo.f:PRINT) is Dummy arg, Ref, Actual arg
          in (demo.f:PRINTIT) is Dummy arg, Ref, Actual arg

*** Vars/Arrays:

COUNT : I*4 : bytes 0-3 of common /BK2/
          in (demo.f:PRINT) is Ref
CURITEM : R*4 : local
          in (demo.f:DIPSTAT) is Dummy arg, Indeterminate, Actual arg
DIMS (3) : R*8 : local
          in (demo.f:READNAME) is Dummy arg, Set
I : I*4 : local
          in (demo.f:PROCDAT) is Ref, Set
INUIT : I*4 : local
          in (demo.f:PRINTIT) is Set, Actual argINUSE : I*2 : bytes 0-1 of
common /BLOCK/
          in (demo.f:PRINTIT) is Ref
ISTAT : I*4 : local
          in (demo.f:DIPSTAT) is Dummy arg, Set
IUNIT : I*4 : local
          in (demo.f:PROCDAT) is Ref, Actual arg
          in (demo.f:PRINT) is Dummy arg, Ref, Actual arg
          in (demo.f:PRINTIT) is Dummy arg, Ref
NAME : CHAR*(*) : local
          in (demo.f:READNAME) is Dummy arg, Set
PUNIT : I*4 : local
          in (demo.f:PROCDAT) is Unused
STATUS : I*4 : bytes 2-5 of common /BLOCK/
          in (demo.f:PRINTIT) is Set
UNIT : I*4 : local
          in (demo.f:GETUNIT) is Dummy arg, Set
UNIT1 : I*4 : local
          in (demo.f:GETUNIT) is Dummy arg, Ref
```

E.6 Tabular Cross Reference

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

```
*** Subroutines:
```

					/---Calls---\	/----------References----------\		
Name	Class	Type	Definition	Arguments	Line-Subprog	Subprog	File	Line
DIPSTAT	subroutine		demo.f line 48	1:(I*4 S) 2:(R*4 R)	49-PRINT	PRINTIT	demo.f	42
GETUNIT	subroutine		demo.f line 14	1:(I*4 S) 2:(I*4 R)		PROCDAT PRINTIT	demo.f demo.f	6 43
PRINT	subroutine		demo.f line 33	1:(struct ITEM R) 2:(I*4 R)	35-PRINTIT	PROCDAT SETTYPE DIPSTAT	demo.f demo.f demo.f	9 30 49
PRINTIT	subroutine		demo.f line 38	1:(I*4 R) 2:(struct ITEM R)	42-DIPSTAT 43-GETUNIT	PRINT	demo.f	35
PROCDAT	program		demo.f line 2		6-GETUNIT 7-READNAME 8-SETTYPE 9-PRINT			
READNAME	subroutine		demo.f line 19	1:(CHAR*(*) S) 2:(R*8 array S)		PROCDAT	demo.f	7
SETTYPE	subroutine		demo.f line 27	1:(struct ITEM RS)	30-PRINT	PROCDAT	demo.f	8

```
*** Common blocks:
```

Name	Size	Members	Consistency	References
BK2	8	COUNT TIME	model same	PROCDAT SETTYPE PRINT PRINTIT
BLOCK	6	INUSE STATUS	model same	PROCDAT READNAME SETTYPE PRINT PRINTIT

```
*** Structures:
                /---------------------------------------Fields------------------------------------------------\
Name      Size    Field       Type              Kind  Attributes  Subprogram    File             References
------------------------------------------------------------------------------------------------------------------
ITEM      26      NAME        CHAR*10                              PROCDAT       demo.f (demo.inc)3-D     7-SA
                                                                   SETTYPE       demo.f (demo.inc)3-D
                                                                   PRINT         demo.f (demo.inc)3-D
                                                                   PRINTIT       demo.f (demo.inc)3-D
------------------------------------------------------------------------------------------------------------------
                  TYPE        I*4                                  PROCDAT       demo.f (demo.inc)4-D
                                                                   SETTYPE       demo.f (demo.inc)4-D      29-S      30-R
                                                                   PRINT         demo.f (demo.inc)4-D      35-R
                                                                   PRINTIT       demo.f (demo.inc)4-D      45-R
------------------------------------------------------------------------------------------------------------------
                  DIMENSIONS  R*4 (3)                              PROCDAT       demo.f (demo.inc)5-D      7-SA
                                                                   SETTYPE       demo.f (demo.inc)5-D     29-R
                                                                   PRINT         demo.f (demo.inc)5-D
                                                                   PRINTIT       demo.f (demo.inc)5-D
------------------------------------------------------------------------------------------------------------------

*** Vars/Arrays:

Name        Type              Kind  Attributes       Subprogram    File        References
------------------------------------------------------------------------------------------------------------------
COUNT       I*4                     bytes 0-3 of     PRINT         demo.f (demo.inc)9-D   (demo.inc)11-D   35-R
                                    common /BK2/
------------------------------------------------------------------------------------------------------------------
CURITEM     struct ITEM             local            PROCDAT       demo.f (demo.inc)7-D      7-SA      7-SA     8-RSA
                                                                                  9-RA
                                                     SETTYPE       demo.f      27-P     (demo.inc)7-D    29-S
                                                                               29-R      30-R      30-RA
                                                     PRINT         demo.f      33-P     (demo.inc)7-D    35-R
                                                                               35-RA
                                                     PRINTIT       demo.f      38-P     (demo.inc)7-D    42-RA
                                                                               45-R
------------------------------------------------------------------------------------------------------------------
CURITEM     R*4                     local            DIPSTAT       demo.f      48-P     49-XA
------------------------------------------------------------------------------------------------------------------
DIMS        R*8 (3)                 local            READNAME      demo.f      19-P     23-D      24-S
------------------------------------------------------------------------------------------------------------------
I           I*4                     local            PROCDAT       demo.f       5-RS     6-R
------------------------------------------------------------------------------------------------------------------
INUIT       I*4                     local            PRINTIT       demo.f      43-SA
------------------------------------------------------------------------------------------------------------------
INUSE       I*2                     bytes 0-1 of     PRINTIT       demo.f (demo.inc)9-D   (demo.inc)10-D   40-R
                                    common /BLOCK/
------------------------------------------------------------------------------------------------------------------
ISTAT       I*4                     local            DIPSTAT       demo.f      48-P     49-S
```

IUNIT	I*4	local		PROCDAT	demo.f	3-D	6-RA	9-RA	11-R
				PRINT	demo.f	33-P	35-RA		
				PRINTIT	demo.f	38-P	40-R	45-R	
NAME	CHAR*(*)	local		READNAME	demo.f	19-P	20-D	24-S	
PUNIT	I*4	local		PROCDAT	demo.f	3-D			
STATUS	I*4	bytes 2-5 of common /BLOCK/		PRINTIT	demo.f (demo.inc)9-D	(demo.inc)10-D	41-S		
UNIT	I*4	local		GETUNIT	demo.f	14-P	15-D	16-S	
UNIT1	I*4	local		GETUNIT	demo.f	14-P	15-D	16-R	

```
----------------- LEGEND ------------------

         A - actual argument
         B - used as an assumed array bound
         C - Associated
         c - Loop Counter
         D - declaration
         E - equivalenced
         F - statement function dummy argument
         G - used as a label in a goto statement
         I - initialized
         i - initialized indirectly
         L - used as a label in an assign statement
         M - allocated
         N - nullified
         O - optional dummy argument
         P - dummy argument
         R - referenced
         S - set
         X - usage cannot be determined
         Z - deallocated

    --------------------------------------------
```

Appendix F

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Diagnostic Messages

F.1 Format

FortranLint 's diagnostic messages are defined in a text file named **flint.err**. This file contains one message per line in the following format:

> ### XX *Text*

where ### is a three-digit message number
XX is a two-letter diagnostic code
Text is the message text

Example:

> 157 SE no matching "(".

If the message number is less than three digits long, it is right-justified in a three-column field.

The first letter of the diagnostic code specifies a basic error category as follows:

Letter	Type of problem
S	Syntax
U	Data usage
I	Call interface
P	Portability
M	Implicit typing
O	Overflow (limit exceeded)

The second letter of the diagnostic code specifies a severity level as follows:

Letter	Type of problem
E	Error
W	Warning
F	Hint (FYI)

Lines that start with an "**I**" are not diagnostic messages. These lines contain information used during portability checking.

F.1 Modifying the flint.err file

NOTE: Modification of the flint.err file is <u>not</u> recommended. Rather, a user should contact Cleanscape to discuss the merits of changing the severity or text content of a Flint analysis message. However, in certain circumstances a customer may decide modification is appropriate for their environment.

In such case, it is recommended that the original flint.err file be saved first as flint.err.orig. Note that any changes would be lost upon installing a new version of Flint obtained from Cleanscape.

If it is decided that the severity of a Flint error message is inappropriate, or the error text is not descriptive enough for a particular application, the flint.err file may be modified using a standard text editor, paying careful attention to existing format.

Appendix G

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Performance

G.1 Disk Space

G.1.1 Program Size

On most systems, FortranLint requires 1 MB to 6 MB of disk space for the package itself. Additional space is required during analysis (see below). On Windows systems or Unix/Linux systems with the GUI option, the size may be up to 20 MB.

G.1.2 Temporary Files

FortranLint generates temporary files during processing. These files are auto-matically deleted upon program termination, including aborts.

The library function **tempnam( )** is used to obtain names for the temporary files. On most systems, the environment variable **TMPDIR** may be used to control the directory used by **tempnam( )**; if **TMPDIR** is not set, **tempnam( )** normally uses **/usr/tmp** or **/tmp**. For additional information, see the UNIX "man" page for **tempnam( ).**

Under VMS, temporary files are placed in the directory specified by **SYS$SCRATCH**.

Generally speaking, temporary files will require 2 MB (or more) of disk space for every 10,000 lines of source code. Cross-reference tables and call trees will increase the amount of disk space required.

Appendix H

❖❖❖
❖❖❖

Xlint Installation, Unix/Linux

H.1 Pre-installation

The Xlint installation procedure assumes that FortranLint has already been installed and activated. If FortranLint has not been installed, see the instructions in Appendix A.

Note: The FortranLint ⁄ Xlint "installation directory" mentioned in the following sections is the directory that contains the FortranLint ⁄ Xlint support files (for example, **flint.err** and **flint.cfg**).

H.2 Installation Procedure

1.   Log in as system manager.

2.   Go to the FortranLint ⁄ Xlint installation directory. The following Xlint support files should already be present:

        xlint   # Xlint executable
        XLint   # Xlint resource file
        demo.fdb   # demo database generated from demo.f

3.   Modify the user configuration for each Xlint user as follows:

(a)   Set the environment variable **XLINTHOST** to the host name of the system where the Xlint license manager will be running. (To obtain the host name, execute the UNIX command **hostname** on the server.)

(b)   Set the environment variable XLINTPATH to a full path for the directory which contains the user's own FORTRAN source files.

(c)   Set the environment variable XLINTHOME to a full path for the Xlint installation directory.

For example, if the user is using **csh**, use commands of the form:

        setenv XLINTHOST nodename
        setenv XLINTPATH source_*path*
        setenv XLINTHOME installation_*directory*
If the user is using **sh**, use commands of the form:

XLINTHOST=*nodename*; export XLINTHOST
XLINTPATH=*source_path*; export XLINTPATH
XLINTHOME=*installation_directory*; export XLINTHOME

Note: There should no white space on either side of the "equals" sign.

For other shells, substitute the appropriate commands.

4.   Add $XLINTHOME to the user's search path. This step can be omitted if $FLINTHOME points to the same directory as $XLINTHOME and $FLINTHOME has been already been added to the search path.

For **csh** users, use the command:

set path=($path $XLINTHOME)

For **sh** users, use the command:

PATH=$XLINTHOME:$PATH

To make the changes permanent, add the new command to the appropriate login scripts. For example, for **csh** users, modify "**.cshrc**".

5.   Optional: The Xlint package includes a utility program **flpatch** that can be used to patch the Xlint installation directory and server name directly into the **xlint** executable.

To patch the executable, use commands of the form:

flpatch xlint home *directory*
flpatch xlint host   *hostname*

Replace *directory* with the Xlint installation directory, and *hostname* with the host name for the system that will be running the Xlint license manager.

Note: The **install_flint** shell script runs **flpatch** automatically. **flpatch** therefore should be needed only if one of these parameters changes.

6.   Copy the XLint resource file (XLint) to the appropriate directory or directories.

A copy of this file should be placed in the home directory for each Xlint user. By default, Xlint uses this copy. Users may specify alternate versions on the Xlint command line; for additional information, see section 15.3.

Alternatively, users may set the standard environment variable **XAPPL**-**RESDIR** or use the standard **app**-**defaults** directory. For additional information, see the operating system vendor's "X" documentation.

7.   Users are now ready to activate Xlint.

H.3 Activation Procedure

Every Xlint license must be assigned a unique authorization number (activation key) before the package will run.

1.   To proceed, execute the following command:

        xlint activate

The software will provide users with a server code, and it will prompt them to call Cleanscape for activation. Cleanscape will use the server code to generate a unique authorization number for the software.

2.   After an activation key is obtained, execute the command:

        xlint activate

again, and enter the activation key when prompted.

3.   If the license manager process **iptlmd** has not already been started, users will need to execute the command:

        startup

from the installation directory. Note that a single **iptlmd** process will allow both FortranLint and Xlint to run.

Users will need to run **startup** every time they reboot the system or kill the license manager. To avoid this step, add the **startup** command to the appropriate system boot script.

4.   The license manager daemon requires a three minute period after being started for initialization. When the three minutes are up, execute the command:

        xlint

Xlint is now ready for use.

Note: For license manager options, see Appendix C.

Appendix **I**

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖

Xlint Installation Under VMS

I.1 Pre-installation

The Xlint installation procedure assumes that FortranLint has already been installed and activated. If FortranLint has not been installed, see the instructions in Appendix B.

Note: The FortranLint ∕ Xlint "installation directory" mentioned in the following sections is the directory that contains the FortranLint ∕ Xlint support files (for example, **flint.err** and **flint.cfg**).

I.2 Installation Procedure

1.   Log in as system manager.

2.   Go to the FortranLint ∕ Xlint installation directory. The following Xlint support files should already be present:

            XLINT.EXE   ! Xlint executable file
            XLINT.DAT   ! Xlint resource file
            DEMO.FDB   ! demo database file for demo.for

3.   Modify the user configuration for each Xlint user as follows:

    (a)   If the Xlint license manager is installed on a DECNET server, set the logical **XLINTHOST** to the node name for the server. Otherwise, set **XLINTHOST** to "**NO_DECNET**".

            Note: To obtain the node name, execute the command "show logical **SYS$NODE**" on the server. Discard any "colon" characters.

    (b)   Set the logical XLINTPATH to a full path for the directory which contains the user's own FORTRAN source files.

    (c)   Set the logical **XLINTHOME** to a full path for the Xlint installation directory.

(d)   Set the logical **XLINT** to a full pathname for the executable file
         XLINT.EXE in the installation directory.

Add the new commands to the appropriate **login.com** files.

Example:

            define XLINTHOST "*nodename*"
            define XLINTPATH [*source path*]
            define XLINTHOME [*installation_directory*]
            XLINT :== $XLINTHOME:XLINT.EXE

4.   Optional: The FortranLint package includes a utility program named
         FLPATCH.EXE that can be used to patch the Xlint installation directory
         and server node name directly into the Xlint executable file.

         To patch Xlint, use commands of the form:

            FLPATCH XLINT.EXE HOME disk:[*directory_path*]
            FLPATCH XLINT.EXE HOST nodename

         disk:[*directory_path*] should specify the Xlint installation directory. *nodename*
         should be the appropriate node name (or "NO_DECNET"), as explained
         in step 3.

5.   Copy the resource file (**XLINT.DAT**) to the appropriate directory or
         directories.

         Two logicals are used:

         DECW$SYSTEM_DEFAULT   -- System directory (same for all users)
         DECW$USER_ DEFAULTS   -- Per-user directory

         To install a copy of **XLINT.DAT** for system-wide use, place it in the
         directory specified by **DECW$SYSTEM_DEFAULTS**. To install a copy
         of **XLINT.DAT** for use by an individual user, place it in the directory
         specified by **DECW$USER_DEFAULTS** for that user.

         Note that users may an alternate resource file on the Xlint command line;
         for additional information, see section 15.3.

6.   Users are now ready to activate Xlint.

I.3 Activation Procedure

Every Xlint license must be assigned a unique authorization number (activation key) before the package will run.

1.   To proceed, execute the following command:

        FLINT /LICENSE=ACTIVATE

The software will provide users with a server code, and it will prompt them to call Cleanscape for activation. Cleanscape will then use this information to provide them with a unique authorization number needed to run the software.

2.   After an activation key is obtained, execute the command:

        FLINT /LICENSE=ACTIVATE

again, and enter the activation code when prompted.

3.   If the license manager (**iptlmd)** hasn't already been started, users will need to execute the command:

        @FLINTHOME:STARTUP.COM

This will start the license daemon. Note that a single **iptlmd** process will allow both FortranLint and Xlint to run.

Users will need to run **@FLINTHOME:STARTUP** again if they reboot the system or kill the detached process. Alternatively, simply add the **STARTUP** command to the appropriate system startup script.

4.   The detached process requires a three-minute period after being started for initialization. When the three minutes are up, execute the command:

        XLINT

Xlint is now ready for use.

Note: For license manager options, see Appendix C.