

USER'S MANUAL

lint-PLUS[®]

SOURCE CODE ANALYZER

Version 5.0x

Cleanscape Software International

5850 Eubank Blvd. NE, STE B49-180

Albuquerque, NM 87111

Tel: 505-246-0267 Fax: 505-247-2678

E-mail: support@cleanscape.net

lint-PLUS[®] USER'S MANUAL
A SOURCE LEVEL CODE ANALYZER
For 'C' PROGRAMMING
On Windows, Unix/Linux, and VMS SYSTEMS

May 2003

Rev. 5.0x

Note: Licensed users may photocopy for distribution.

Direct comments concerning this manual to:

Cleanscape Software International
5850 Eubank Blvd. NE, B49-180, Albuquerque, NM 87111
Tel: 505-246-0267 Fax: 505-247-2678
E-mail: support@cleanscape.net

Copyright 1987-2007

Cleanscape
NOTICE OF COPYRIGHTS

Copyrighted by Cleanscape as an unpublished work.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

NOTICE OF PROPRIETARY RIGHTS

This manual and the material on which it is recorded are the property of Cleanscape. Its use, reproduction, transfer and/or disclosure to others, in this or any other form, is prohibited except as permitted by a written License Agreement with Cleanscape.

Cleanscape reserves the right to update this document without prior notification.

lint-PLUS is a registered trademark of Cleanscape Software International.
UNIX is a registered trademark of AT&T Bell Laboratories.
DEC, ULTRIX, VAX, and VMS are registered trademarks of Digital Equipment Corp.

1



Introduction

1.1 Overview

lint-PLUS is a programming tool that simplifies the debugging and maintenance of both large and small 'C' programs.

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

- inappropriate arguments passed to functions
- inappropriate library calls
- non-portable code
- type usage conflicts across different modules
- unused variables and dead code

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

2



Getting Started

2.1 Analyzing programs

To run lint-PLUS, use a command of the form:

```
lplus file1.c file2.c file3.c ...
```

where the specified files are 'C' source files.

lint-PLUS will check the source files for syntax errors and other “local” problems. The results will be printed to the standard-output stream.

To check for “global” (inter-module) problems as well as “local” problems, add the option switch “-g”:

```
lplus -g file1.c file2.c file3.c ...
```

Wildcards are allowed. For example, the following command will process all of the 'C' source files in the current directory:

```
lplus -g *.c
```

If “**lplus**” is executed without any arguments, lint-PLUS will print a help screen:

```
lplus
```

2.2 Creating an output file

To send lint-PLUS output to a text file, use a command similar to the following:

lplus file1.c file2.c ... > outfile under UNIX
or
lplus /OUTPUT=outfile file1.c file2.c ... under VMS

where *outfile* is the desired output-file name. lint-PLUS will analyze the specified source files and print the results to the specified output file.

2.3 Command files (overview)

Command-line arguments may be specified indirectly. If “**foo.txt**” is a text file containing option switches or filenames, the following command will add the contents of “**foo.txt**” to the lint-PLUS argument list:

```
lplus @foo.txt ...
```

Text files which are used this way are called “command files”. For additional information, see section 9.2.

2.4 Options (overview)

lint-PLUS accepts a number of command-line option switches, which may be used to enable or disable various features.

For example, the “**-g**” switch is used to enable “global” checking and the “**-x**” switch is used to produce a cross-reference. The following command enables both options:

```
lplus -gx file1.c file2.c file3.c ...
```

lint-PLUS supports “word” switches as well as “letter” switches. For example, the following commands also enable the global-checking and cross-reference options:

```
lplus -global -xref file1.c file2.c ...      under UNIX  
or  
lplus /GLOBAL /XREF file1.c file2.c ...      under VMS
```

For additional information, see chapters 3 through 5.

2.5 UNIX environment variables

The user may set various environment variables to change the behavior of lint-PLUS under UNIX.

For example, the following **cs** command will add the directories “/usr/local/include” and “/prod” to the default “local” include-file search path:

```
setenv LOCINC “/usr/local/include:/prod”
```

UNIX lint-PLUS recognizes the following environment variables:

Variable	Specifies
=====	=====
INCLUDE	same as LOCINC (*)
CSHOST	see appendix A
CSHOME	lint-PLUS installation directory
LINTCFG	user's own lint-PLUS directories (*)
LOCINC	“local” include-file directories (*)
STDINC	“standard” include-file directories (*)

(*) Variable specifies a directory or list of directories in the following format:

```
directory-path
or directory-path:....:directory-path
```

Directory paths must be separated by colons.

CSHOST and **CSHOME** must be defined. The other four environment variables are optional.

For additional information on **CSHOST** and **CSHOME**, see appendix A.

For additional information on the other four variables, see chapter 7.

2.6 VMS logicals

The user may set various logicals to change the behavior of lint-PLUS under VMS.

For example, the following DCL command will add the directories “[**prod**]” and “[**test**]” to the “local” include-file search path:

```
$ define C$INCLUDE [proj],[test]
```

VMS lint-PLUS recognizes the following logicals:

Logical	Specifies
=====	=====
C\$INCLUDE	“local” include-file directories (*)
CSIHST	see appendix B
IP\$Lint	lint-PLUS installation directory
LINT\$CFG	user's own lint-PLUS directories (*)
SY\$Library	standard include-file directories (*)
VAXC\$INCLUDE	ditto (*)

(*) Logical specifies a directory or list of directories in the following format:

```
directory-specifier  
or directory-specifier,...,directory-specifier
```

Directory specifiers must be separated by commas.

CSIHST and **IP\$Lint** must be defined. The other four logicals are optional.

For additional information on **CSIHST** and **IP\$Lint**, see appendix B.

For additional information on the other four logicals, see chapter 7.

3



UNIX option format

3.1 Overview

lint-PLUS accepts a number of command-line option switches, which may be used to enable or disable various features.

Option switches may be specified anywhere in the command line. For example, the following two commands are equivalent:

```
lplus -n foo.c bar.c
lplus foo.c -n bar.c
```

Options may also be specified in “command files”. For additional information on command files, see section 9.2.

For a list of options, see chapter 5.

3.2 "Letter" and "word" switches

lint-PLUS supports both “letter” and “word” option switches.

For example, “-g” is a “letter” switch that enables global (inter-module) checking and “-global” is a “word” switch which does the same thing:

```
lplus -g      foo.c
lplus -global foo.c
```

“Letter” switches may be combined. For example, the following two commands are equivalent:

```
lplus -g -s -x foo.c  
lplus -gsx    foo.c
```

Options may be specified in upper or lower case. For example, the following two commands are equivalent:

```
lplus -global    foo.c
lplus -GLOBAL   foo.c
```

“Word” option names may be abbreviated to a minimum of four characters. For example, the following two commands are equivalent:

```
lplus -standard=/proj/include foo.c
lplus -stan=/proj/include    foo.c
```

Note: If a non-unique abbreviation is used, lint-PLUS will print an error message.

3.3 Option switch arguments

If an option switch accepts arguments, they should be specified as follows:

```
-option=argument
or -option=argument,...,argument
```

For example:

```
lplus -disable=33
lplus -disable=33,42,57
```

The following four “letter” switches are special cases:

```
-D    Defines    a preprocessor symbol
-U    Un-defines a preprocessor symbol
-I    (upper-case eye) Adds a local include-file directory
-S    Adds a standard include-file directory
```

The “equals” sign is optional for these four switches. For example, the following two commands are equivalent:

```
lplus -DDEBUG -I/proj/include
lplus -D=DEBUG -I=/proj/include
```

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

```
lplus -d33 -d42 -d57
lplus -d33,42 -d57
lplus -d33,42,57
```

“Letter” switches are case-sensitive. “Word” switches may be specified in upper or lower case.

3.4 Overriding options (UNIX)

To cancel an option switch, or to override a default setting, use a switch of the form “dash dash option”. For example, the following two commands are equivalent:

```
lplus -g --g foo.c  
lplus foo.c
```

This feature may be used to override switches set by a command file. For example, the following command uses the switches set by “foo.txt”, but disables the “port” option, if it is set:

```
lplus @foo.txt --port ...
```

Note: If arguments are specified for an option switch, they are discarded when the option is cancelled. For example, the following two commands are equivalent:

```
lplus -lxxx,yyy --l -lzzz foo.c  
lplus -lzzz foo.c
```


4



VMS Option Format

4.1 Overview

lint-PLUS accepts a number of command-line option switches, which may be used to enable or disable various features.

Option switches may be specified anywhere in the command line. For example, the following two commands are equivalent:

```
lplus -n foo.c bar.c
lplus foo.c -n bar.c
```

Options may also be specified in “command files”. For additional information on command files, see section 9.2.

For a list of options, see chapter 5.

4.2 "Letter" and "word" switches

lint-PLUS supports both “letter” and “word” option switches. “Letter” switches start with a dash and “word” switches start with a slash (/).

For example, “-g” is a “letter” switch that enables global (inter-module) checking and “/global” is a “word” switch which does the same thing:

```
lplus -g      foo.c
lplus /global foo.c
```

“Letter” switches may be combined. For example, the following two commands are equivalent:

```
lplus -g -s -x foo.c  
lplus -gsx    foo.c
```


Options may be specified in upper or lower case. For example, the following two commands are equivalent:

```
lplus /global    foo.c
lplus /GLOBAL   foo.c
```

“Word” option names may be abbreviated to a minimum of four characters. For example, the following two commands are equivalent:

```
lplus /standard=[-.incldir] foo.c
lplus /stan=[-.incldir]    foo.c
```

Note: If a non-unique abbreviation is used, lint-PLUS will print an error message.

4.3 Option switch arguments

If a “word” option switch accepts arguments, they should be specified as follows:

```
/option=argument
or /option=(argument,...,argument)
```

For example:

```
lplus /disable=33
lplus /disable=(33,42,57)
```

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

```
lplus /disable=33 /disable=42 /disable=57
lplus /disable=(33,42) /disable=57
lplus /disable=(33,42,57)
```

“Letter” switches are similar to “word” switches. For example, the following two commands are equivalent:

```
lplus /disable=33
lplus -d=33
```

However, “letter” switches don’t accept multiple arguments. To specify extra parameters for a “letter” option, use extra copies of the “letter” switch:

```
-option=arg1 -option=arg2 ...
```

For example, the following two commands are equivalent:

```
lplus /disable=(33,42,57)
lplus -d=33 -d=42 -d=57
```

The following four “letter” switches are special cases:

-D	Defines	a preprocessor symbol
-U	Un-defines	a preprocessor symbol
-I	(upper-case eye)	Adds a local include-file directory
-S	Adds a standard	include-file directory

The “equals” sign is optional for these four switches. For example, the following two commands are equivalent:

```
lplus -DDEBUG -I/proj/include
lplus -D=DEBUG -I=/proj/include
```

“Letter” switches are case-sensitive. “Word” switches may be specified in upper or lower case.

4.4 Overriding options (VMS)

To cancel a “word” option switch, or to override a default “word” option setting, use a switch of the form “/NO...”. For example, the following two commands are equivalent:

```
lplus /global /noglobal foo.c
lplus foo.c
```

To cancel a “letter” option switch, or to override a default “letter” option setting, use a switch of the form “dash dash option”. For example, the following two commands are equivalent:

```
lplus -g --g foo.c
lplus foo.c
```

This feature may be used to override switches set by a command file. For example, the following command uses the switches set by “foo.txt”, but disables the “port” option, if it is set:

```
lplus @foo.txt /noport ...
```

Note: If arguments are specified for an option switch, they are discarded when the option is cancelled. For example, the following two commands are equivalent:

```
lplus /local=(xxx,yyy) /nolocal /local=zzz foo.c
lplus /local=zzz foo.c
```


5



List of options

5.1 Overview

Section 5.2 summarizes the lint-PLUS options and section 5.3 contains a complete list, including sub-options.

For an explanation of UNIX option formats, see chapter 3.

For an explanation of VMS option formats, see chapter 4.

5.2 Summary of options

Option name		Purpose
ALL		Combines several options
ARCHAIC		Allows “archaic” initialization statements
BEEP		Controls audible output
BRIEF		Skips repeated local error messages
DASHES		Adds separators before error messages
DEFINE	(-D)	Defines a preprocessor symbol
DISABLE	(-d)	Disables specific local error messages (*)
ENABLE	(-e)	Enables specific local error messages (*)
FULLINIT		Reports incomplete initialization
GLOBAL	(-g)	Enables global (inter-module) checking
GRAPHICS		Changes the call-tree graphics characters
HTRREE		Generates “include file” trees
LIB		Checks calls to system library routines
LIST	(-l)	(Lower-case ell) Generates source code listings
LOCAL	(-I)	(Upper-case eye) Adds local include-file directories
MAXERROR		Sets max. number of local errors per module
MAXFATAL		Sets max. number of fatal errors per run
MAXGSE		Sets max. number of global errors per symbol
NITPICK	(-n)	Enables “FYI” (informational) messages
ODD		Reports valid but unusual constructs
PAGE		Controls pagination
PORT	(-p)	Controls portability checking
PROTO		Generates ‘C’ prototypes or FORTRAN “shells”
RESULTS		Checks for unused function-call results
SIGN		Controls signed/unsigned checking
SILENT		Disables progress messages
STANDARD	(-S)	Adds standard include-file directories
STATISTICS	(-s)	Enables statistical reports
STRICT		Adds extra checks to “global mode”
SYSTEM		Selects a target system
TREE	(-t)	Generates call trees
UNDEFINE	(-U)	Un-defines a preprocessor symbol
UNUSED		Reports unused local variables
VERBOSE	(-v)	Controls message verbosity
WARNING	(-w)	Enables warning messages
XREF	(-x)	Generates a cross-reference

(*) “-letter” version of option is not supported under VMS. Use the “/word” version instead.

5.3 Option list

Note: “Letter” switches are case-sensitive. “Word” switches and sub-option names are not case-sensitive.

Square brackets are used to indicate optional text. For example, “graph[ics]” means “graph or graphics”.

“Local” errors are localized problems such as syntax errors, uninitialized variables, etc. “Global” errors are inter-module or call-level problems such as inconsistent argument lists, variable data-type conflicts, etc.

ALL Combines the FULLINIT, NITPICK, UNUSED and VERBOSE options.

UNIX syntax: -all or --all (to cancel)

VMS syntax: /all or /noall (to cancel)

Related options: FULLINIT, NITPICK, UNUSED, VERBOSE

ARCHAIC Controls handling of “archaic” initialization statements.

Some older 'C' programs use initialization statements that omit the “equals” sign; for example, “int x 5” might be used instead of “int x = 5”.

lint-PLUS normally treats these statements as errors. ARCHAIC downgrades the error messages to warnings.

UNIX syntax: -arch[aic] or --arch[aic] (to cancel)

VMS syntax: /arch[aic] or /noarch[aic] (to cancel)

BEEP Controls audible output. If BEEP is selected, and if output is being sent to a console, lint-PLUS will generate a beep when a “local” message is printed. The following sub-options are supported:

BEEP=F	Beep for “fatal” messages only.
BEEP=E	Beep for “error” or “fatal” messages.
BEEP=W	Beep for “warning”, “error”, or “fatal” messages.

UNIX syntax: -beep or -beep=*option* or --beep (to cancel)

VMS syntax: /beep or /beep=*option* or /nobeep (to cancel)

BRIEF

Skips repeated local error messages.

If a local error message is printed three times for a given source file, lint-PLUS normally ignores subsequent occurrences of the same error in the same file.

BRIEF may be used to modify this behavior. To change the limit of three to a different value, use “BRIEF=*number*”, where *number* is the desired value.

To remove the limit entirely, override BRIEF (using “--brief” under UNIX or “/nobrief” under VMS). To restore the default mode of operation, use “-brief” or “/brief”.

UNIX syntax: -brief or -brief=*number* or --brief (to cancel)

VMS syntax: /brief or /brief=*number* or /nobrief (to cancel)

DASHES

If DASHES is selected, lint-PLUS will insert a separator (i.e., a line of dashes) before each local error message.

UNIX syntax: -dash[es] or --dash[es] (to cancel)

VMS syntax: /dash[es] or /nodash[es] (to cancel)

DEFINE

Defines a preprocessor symbol. Two formats are supported:

DEFINE=*symbol* Asserts that the specified symbol is defined.

DEFINE=*symbol:string* Defines the specified symbol using the specified value.

To erase a definition which was set previously, use the UNDEFINE option.

UNIX syntax: -D*symbol* or -define=*symbol* or
 -D*symbol:string* or -define=*symbol:string*

VMS syntax: -D“*symbol*” or /define=“*symbol*” or
 -D“*symbol:string*” or /define=“*symbol:string*”

Note: Under VMS, arguments to “-D” or “/define” should be quoted.

Related options: UNDEFINE

DISABLE

Disables one or more local error messages.

To disable specific messages, pass the associated message numbers to DISABLE. For example:

lplus -disable=10,45,74	under UNIX
lplus /disable=(10,45,74)	under VMS

Note: DISABLE does not affect “global” errors or fatal errors (such as missing include files).

To disable all non-fatal local error messages, use “DISABLE=all”.
To enable messages which have been disabled, use the ENABLE option.

UNIX syntax: -disable=msg# or
 -disable=msg#,msg#,... or -disable=all

Can also use “-d=msg#,...” or “-d=all”

VMS syntax: /disable=msg# or
 /disable=(msg#,msg#,...) or /disable=all

Note: The “letter” switch (-d) is not supported under VMS.

Related options: ENABLE

ENABLE

May be used to enable error messages which were disabled previously by DISABLE.

To enable specific messages, pass the associated message numbers to ENABLE. For example:

lplus -enable=10,45,74	under UNIX
lplus /enable=(10,45,74)	under VMS

To enable all local error messages, use “ENABLE=all”.

UNIX syntax: -enable=msg# or
 -enable=msg#,msg#,... or -enable=all

Can also use “-e=msg#,...” or “-e=all”

VMS syntax: /enable=msg# or
 /enable=(msg#,msg#,...) or /enable=all

Note: The “letter” switch (-e) is not supported under VMS.

Related options: DISABLE

FULLINIT

Reports incomplete initialization of arrays or structures.

Note: The ALL option sets FULLINIT.

UNIX syntax: -full[init] or --full[init] (to cancel)

VMS syntax: /full[init] or /nofull[init] (to cancel)

Related options: ALL

GLOBAL

Enables global (inter-module) checking. For additional information on global checking, see section 6.4.

UNIX syntax: -global or --global (to cancel)

Can also use “-g” or “--g”

VMS syntax: /global or /noglobal (to cancel)

Can also use “-g” or “--g”

Related options: LIB, MAXGSE, PROTO, SIGN, STRICT, TREE, XREF

GRAPHICS

Changes the graphics characters used to print call trees.

GRAPHICS takes five arguments. The arguments are ASCII character codes expressed as hexadecimal numbers. The five character codes are interpreted as follows:

- 1st character - used to draw horizontal lines
- 2nd character - used to draw vertical lines
- 3rd character - used to draw “T” intersections
- 4th character - used to draw “I-” intersections
- 5th character - used to draw “L”-shaped corners

The default settings are 2D, 7C, 2B, 2B, and 2B .

For additional information on call trees, see section 6.5.

UNIX syntax: -graph[ics]=#,#,#,#,# or --graph[ics] (to cancel)

VMS syntax: /graph[ics]=(#,#,#,#,#) or /nograph[ics] (to cancel)

Related options: TREE

HTREE

Generates “include file” trees. The following sub-options are supported:

HTREE=clip	Equivalent to “HTREE=clip:72”.
HTREE=clip: <i>number</i>	By default, HTREE clips include-file trees at column 72. Extra “continuation” trees are generated to display branches which are cut off. To change the default column number, use “HTREE=clip: <i>number</i> ”. <i>number</i> should be greater than 30. To restore the default mode of operation, use “HTREE=clip”.
HTREE=noclip	To disable clipping entirely, use “noclip”.
HTREE=file: <i>filename</i>	If this option is used, include-file trees will be printed to the specified file. To cancel this sub-option, use “HTREE=nofile”.
HTREE=merge	By default, HTREE produces a separate include-file tree for every source file. If the “merge” sub-option is used, HTREE will produce a single tree for all source files. Note: Merged trees will necessarily omit some information. To cancel this sub-option, use “HTREE=nomerge”.
HTREE=path	HTREE normally displays include-file names without path components. If “path” is used, HTREE will display pathnames (where applicable). To cancel this sub-option, use “HTREE=nopath”.
HTREE=nosquash	“HTREE=nosquash” adds extra white space to include-file trees. This improves readability, but increases the size of the output. To restore the default mode of operation, use “HTREE=squash”.

HTREE=notrim To reduce the size of the output, HTREE normally merges redundant sub-trees. If “notrim” is used, HTREE will generate sub-trees whether or not they are redundant.

Note: “notrim” may increase the size of the output significantly.

To restore the default mode of operation, use “HTREE=trim”.

Sub-options may be combined. For example:

lplus -htree=clip:70,squash	under UNIX
lplus /htree=(clip:70,squash)	under VMS

For additional information on “include file” trees, see section 6.6.

UNIX syntax: -htree or
 -htree=*option*,... or --htree (to cancel)

Can also use “-h” or “-h=*option*,...” or “--h”

VMS syntax: /htree or
 /htree=(*option*,...) or /nohtree (to cancel)

Can also use “-h” or “--h”

LIB

Adds system-library checks to “global mode”. This option is on, by default.

The lint-PLUS package includes a 'C' source file named “**syslib.c**” that defines a number of library function prototypes. If the GLOBAL option is used, lint-PLUS normally checks function calls against the prototypes in “**syslib.c**”.

To disable the “**syslib.c**” checks, override LIB (using “--lib” under UNIX or “/nolib” under VMS). To restore the default mode of operation, use “-lib” or “/lib”.

For additional information on global checking, see section 6.4.

For additional information on “**syslib.c**”, see section 9.5.

UNIX syntax: -lib or --lib (to cancel)

VMS syntax: /lib or /nolib (to cancel)

Related options: GLOBAL

LIST

Generates a source-code listing. The following sub-options are supported:

LIST=clip	Equivalent to “LIST=clip:72”.
LIST=clip: <i>number</i>	<p>By default, LIST clips listing output lines at column 72.</p> <p>To clip listing output lines at a different column, use “LIST=clip:<i>number</i>”, where <i>number</i> specifies the column number. <i>number</i> should be greater than 30.</p> <p>To wrap long lines instead of clipping them, use the “wrap” sub-option described below. To disable both clipping and wrapping, use “LIST=nomax”.</p>
LIST=ff	<p>By default, LIST generates its own page breaks; literal formfeeds in the source code are normally ignored.</p> <p>If “ff” is specified, LIST will not generate its own page breaks. Page breaks will be produced if and only if formfeeds are encountered in the source code.</p> <p>To cancel this sub-option, use “LIST=noff”.</p> <p>For additional information on pagination, see the PAGE option.</p>
LIST=include	<p>Listings normally exclude the contents of “include” files. To add the contents of “include” files, use “LIST=include”.</p> <p>To cancel this sub-option, use “LIST=noinclude”.</p> <p>“include” may be abbreviated to “inc”. “noinclude” may be abbreviated to “noinc”.</p>
LIST=nomax	Disables both clipping and wrapping.

LIST=wrap	Equivalent to “LIST=wrap:72”.
LIST=wrap: <i>number</i>	By default, LIST clips long output lines. Maximum line length is controlled by the “clip” sub-option described previously. To wrap output lines at a given column instead of clipping them, use “LIST=wrap: <i>number</i> ”, where <i>number</i> specifies the column number. <i>number</i> should be greater than 30. To disable both clipping and wrapping, use “LIST=nomax”.

Sub-options may be combined. For example:

lplus -list=clip:70,ff	under UNIX
lplus /list=(clip:70,ff)	under VMS

UNIX syntax: -list or
 -list=*option*,... or --list (to cancel)

Can also use “-l” (lower-case ell) or “-l=*option*,...” or “--l”

VMS syntax: /list or
 /list=(*option*,...) or /nolist (to cancel)

Can also use “-l” (lower-case ell) or “--l”

Related options: PAGE

LOCAL

May be used to specify directories that contain “local” (double-quoted) include files, as opposed to “standard” (angle-bracketed) include files. For additional information, see chapter 7.

UNIX syntax: -local=*directory* or
 -local=*dir1,dir2*,... or --local (to cancel)

Can also use “-Idir” (upper-case eye) or “-Idir1,dir2,...” or “--I”

VMS syntax: /local=*directory* or
 /local=(*dir1,dir2*,...) or /nolocal (to cancel)

Can also use “-Idir” (upper-case eye) or “--I”

Related options: STANDARD

MAXERROR

Places a limit on the number of “local” (intra-module) errors to be processed per source file.

MAXERROR takes an integer argument:

MAXERROR=*number*

If this option is used, lint-PLUS will print up to *number* “local” error messages per source file. If an additional “local” error is detected, lint-PLUS will print a “fatal error” message and go on to the next file.

UNIX syntax: -maxerr[or]=*number* or --maxerr[or] (to cancel)

VMS syntax: /maxerr[or]=*number* or /nomaxerr[or] (to cancel)

Related options: MAXFATAL, MAXGSE

MAXFATAL

Places a limit on the number of “fatal” errors to be processed per run.

MAXFATAL takes an integer argument:

MAXFATAL=*number*

A “fatal” error is an error which prevents lint-PLUS from processing a given source file; for example, a missing include file. If MAXFATAL is used, lint-PLUS will allow up to *number* fatal errors per run. If an additional “fatal” error occurs, lint-PLUS will print a final message and terminate the run.

UNIX syntax: -maxfatal=*number* or --maxfatal (to cancel)

VMS syntax: /maxfatal=*number* or /nomaxfatal (to cancel)

Related options: MAXERROR, MAXGSE

MAXGSE

Changes the limit on the maximum number of global errors to be processed per symbol.

MAXGSE takes an integer argument:

MAXGSE=*number*

By default, the GLOBAL option will generate up to ten global error messages per symbol. If MAXGSE is used, the limit will be set to *number*. To remove the limit entirely, use “--maxgse” (under UNIX) or “/nomaxgse” (under VMS).

For additional information on global checking, see section 6.4.

UNIX syntax: -maxgse=*number* or --maxgse (to remove limit)

VMS syntax: /maxgse=*number* or /nomaxgse (to remove limit)

Related options: GLOBAL, MAXERROR, MAXFATAL

NITPICK

Enables “FYI” (informational) messages.

lint-PLUS normally prints error messages for “local” errors and warning messages for less-serious problems. NITPICK adds “FYI” (informational) messages.

Note: The ALL option sets NITPICK.

UNIX syntax: -nitpick or --nitpick (to cancel)

Can also use “-n” or “--n”

VMS syntax: /nitpick or /nonitpick (to cancel)

Can also use “-n” or “--n”

Related options: ALL, WARNING

ODD

Reports valid but unusual constructs, such as passing structures by value.

Note: If the PORT option is turned on, some valid but unusual constructs will be reported whether or not ODD is used.

UNIX syntax: -odd or --odd (to cancel)

VMS syntax: /odd or /noodd (to cancel)

PAGE

Controls pagination.

If the LIST option is selected, lint-PLUS paginates listing output (including “local” error messages, if any). PAGE may be used to set the page length.

PAGE takes an integer argument, which may range from 20 to 99:

PAGE=*number*

lint-PLUS uses a page length of 66 lines, by default. If PAGE is used, page length will be set to *number*.

To disable pagination, override “PAGE” (using “--page” under UNIX or “/nopage” under VMS). To restore the default mode of operation, use “-page” or “/page”.

Note: The LIST option overrides PAGE. If LIST is used after PAGE on the command line, PAGE is ignored.

UNIX syntax: -page[=*number*] or --page (to disable pagination)

VMS syntax: /page[=*number*] or /nopage (to disable pagination)

Related options: LIST

PORT

Controls portability checking.

If PORT is specified, lint-PLUS checks for general portability issues. (To check for issues related to a particular environment, use the SYSTEM option.)

UNIX syntax: -port or --port (to cancel)

Can also use “-p” or “--p”

VMS syntax: /port or /noport (to cancel)

Can also use “-p” or “--p”

PROTO

Generates ‘C’ prototypes or FORTRAN shells. The following sub-options are supported:

PROTO=file:filename Generates ‘C’ prototypes or FORTRAN “shells” for the current set of input files and sends the output to *filename*.

Note: The “file” sub-option is required. If this sub-option is not specified, PROTO generates no output.

PROTO=fortran By default, PROTO generates ‘C’ prototypes. If the “fortran” sub-option is used, PROTO generates FORTRAN “shells” instead.

For additional information on FORTRAN “shells”, see section 6.7.

PROTO=nodef[ined] By default, PROTO generates prototypes for functions that are defined by the current set of input files.

If the “nodefined” sub-option is used, PROTO will generate a prototype for functions which are either defined or prototyped by the input files.

If a function is declared, but not defined or prototyped, PROTO will generate a declaration for the function instead.

Note: The “fortran” and “nodefined” sub-options cannot be used together.

PROTO=def[ined] To cancel the “nodefined” sub-option, use “defined”.

Sub-options may be combined. For example:

<code>lplus -proto=file:foo.txt,fortran</code>	under UNIX
<code>lplus /proto=(file:foo.txt,fortran)</code>	under VMS

Note: PROTO enables “global mode” (i.e., the GLOBAL option).

UNIX syntax: `-proto=option,...` or `--proto` (to cancel)

VMS syntax: `/proto=(option,...)` or `/nopproto` (to cancel)

Related options: GLOBAL

RESULTS

Checks for unused function-call results. If this option is used, lint-PLUS will report function calls which return a value that is not used by the caller.

UNIX syntax: -results or --results (to cancel)

VMS syntax: /results or /noresults (to cancel)

SIGN

Adds signed/unsigned checking to “global mode”.

If SIGN is used, the GLOBAL option will report function-argument sign conflicts. (For additional information on global checking, see section 6.4.)

UNIX syntax: -sign or --sign (to cancel)

VMS syntax: /sign or /nosign (to cancel)

Related options: GLOBAL

SILENT

Disables progress messages. lint-PLUS normally prints a header line for each source file as it is processed. If SILENT is used, the header lines are suppressed.

SILENT does not affect error messages or reports (cross-references, call trees, etc.).

Note: The LIST option overrides SILENT. If LIST is used, SILENT is ignored.

UNIX syntax: -silent or --silent (to cancel)

VMS syntax: /silent or /nosilent (to cancel)

STANDARD

May be used to specify directories that contain “standard” (angle-bracketed) include files, as opposed to “local” (double-quoted) include files. For additional information, see chapter 7.

UNIX syntax: -stan[dard]=*directory* or
-stan[dard]=*dir1,dir2,...* or --stan[dard] (to cancel)

Can also use “-Sdir” or “-Sdir1,dir2,...” or “--S”

VMS syntax: /stan[dard]=*directory* or
/stan[dard]=(*dir1,dir2,...*) or /nostan[dard] (to cancel)

Can also use “-Sdir” or “--S”

Related options: LOCAL

STATISTICS

Enables statistical reports.

If this option is used, lint-PLUS will generate statistical reports, including error summaries, I/O counts, and related information.

STATISTICS supports the following sub-options:

STATISTICS=noglob[al]

STATISTICS=noloc[al]

STATISTICS generates both local (per module) statistics and global (per run) statistics. The “noglobal” sub-option disables the global statistics and the “noloc” sub-option disables the local statistics.

STATISTICS=glob[al]

STATISTICS=loc[al]

To cancel “noglobal”, use “global”.

To cancel “noloc”, use “local”.

Sub-options may be combined. For example:

lplus -stat=global,noloc

lplus /stat=(global,noloc)

under UNIX

under VMS

For additional information, see section 6.9.

UNIX syntax:

-stat[istics] or

-stat[istics]=*option*,... or --stat[istics] (to cancel)

Can also use “-s” or “-s=*option*,...” or --s

VMS syntax:

/stat[istics] or

/stat[istics]=(*option*,...) or /nostat[istics] (to cancel)

Can also use “-s” or “--s”

STRICT

Adds extra checks to “global mode”.

“STRICT=intsize” adds the “intsize” check described below.

“STRICT=proto” adds the “proto” check described below.

If no arguments are specified, STRICT adds both checks.

Note: STRICT enables “global mode” (i.e., the GLOBAL option).

STRICT=intsize If integers, longs and/or shorts are the same size on a given system, “global mode” normally disregards global-variable discrepancies of the following form:

```
extern int foo;    /* source file #1 */
extern long foo;   /* source file #2 */
extern short foo;  /* source file #3 */
```

If “STRICT=intsize” is used, “global mode” will report discrepancies of this type, regardless.

Note (a): This option affects inter-module checking. lint-PLUS will report intra-module conflicts (discrepancies inside a source file) as “local mode” errors, whether or not “intsize” is used.

Note (b): The “portability” option (PORT) turns “STRICT=intsize” on.

To disable this sub-option, use “STRICT=nointsize”.

STRICT=proto If “STRICT=proto” is used, “global mode” will add a section to the output which lists “unprototyped” function calls; i.e., calls to functions which are not prototyped in code preceding the call.

To disable this sub-option, use “STRICT=noproto”.

Sub-options may be combined. For example:

lplus -strict=intsize,noproto	under UNIX
lplus /strict=(intsize,noproto)	under VMS

UNIX syntax: -strict or
 -strict=option,... or --strict (to cancel)

VMS syntax: /strict or
 /strict=(*option*,...) or /nostrict (to cancel)

Related options: GLOBAL

SYSTEM

Selects a target system.

SYSTEM may be used to check for problems related to a particular environment.

The following sub-options are supported:

SYSTEM=host	Assume that 'C' source files are intended for use on the current host system. This is the default setting.
SYSTEM=hpux	Target is an HPUX system.
SYSTEM=none	Don't assume any particular target system.
SYSTEM=sgi	Target is a Silicon Graphics system.
SYSTEM=sparc	Target is SunOS running on a Sparc system.
SYSTEM=sun	Same as "sparc".
SYSTEM=sun3	Target is SunOS running on a Sun-3 system.
SYSTEM=ultrix	Target is an ULTRIX system.
SYSTEM=unix	Target is a generic UNIX system.
SYSTEM=vms	Target is a VMS system.

Note: Target-system names may be specified in upper or lower case.

UNIX syntax: -system=*system-name* or --system (to cancel)

VMS syntax: /system=*system-name* or /nosystem (to cancel)

TREE

Generates function-call trees. The following sub-options are supported:

TREE=clip	Equivalent to “TREE=clip:72”.
TREE=clip: <i>number</i>	By default, TREE clips call trees at column 72. Extra “continuation” trees are generated to display branches which are cut off. To change the default column number, use “TREE=clip: <i>number</i> ”. <i>number</i> should be greater than 30. To restore the default mode of operation, use “TREE=clip”.
TREE=noclip	To disable clipping entirely, use “noclip”.
TREE=file: <i>filename</i>	If “file” is used, call trees will be stored in the specified file. To cancel this sub-option, use “TREE=nofile”.
TREE=squash	To improve readability, TREE normally adds extra white space to call trees. “squash” removes the extra space. To restore the default mode of operation, use “TREE=nosquash”.
TREE=notrim	To reduce the size of the output, TREE normally merges redundant sub-trees. If “notrim” is used, TREE generates sub-trees whether or not they are redundant. Note: “notrim” may increase the size of the output significantly. To restore the default mode of operation, use “TREE=trim”.

Sub-options may be combined. For example:

lplus -tree=clip:70,squash	under UNIX
lplus /tree=(clip:70,squash)	under VMS

Note: TREE enables “global mode” (i.e., the GLOBAL option).

For additional information on call trees, see section 6.5.

UNIX syntax:

-tree	or
-tree= <i>option</i> ,...	or --tree (to cancel)

Can also use “-t” or “-t=*option*,...” or “--t”

VMS syntax: /tree or
 /tree=(option,...) or /notree (to cancel)

Can also use “-t” or “--t”

Related options: GLOBAL

UNDEFINE

May be used to erase a preprocessor symbol defined previously by `DEFINE`.

UNIX syntax: -U*symbol* or -undef[ine]=*symbol*

VMS syntax: -U“symbol” or /undef[ine]=“symbol”

Note: Under VMS, arguments to “-U” or “/undefine” should be quoted.

Related options: DEFINE

UNUSED

Reports unused local variables.

Note: The ALL option sets UNUSED.

UNIX syntax: -unused or --unused (to cancel)

VMS syntax: /unused or /nounused (to cancel)

Related options: ALL

VERBOSE

Controls message verbosity.

Local-error messages are normally one or two lines long. VERBOSE adds background information for most local errors.

The extra text will be printed at most once per source file. If an error occurs more than once in a source file, the extra text will be printed only for the first occurrence of the error.

Note: The ALL option sets VERBOSE.

UNIX syntax: -verb[ose] or --verb[ose] (to cancel)

Can also use “-v” or “--v”

VMS syntax: /verb[ose] or /noverb[ose] (to cancel)

Can also use “-v” or “--v”

Related options: ALL

WARNING

Enables warning messages. This option is on, by default.

lint-PLUS normally prints error messages for “local” errors and warning messages for less-serious problems.

To disable warning messages, override WARNING (using “--warn” under UNIX or “/nowarn” under VMS). To restore the default mode of operation, use “-warn” or “/warn”.

UNIX syntax: -warn[ing] or --warn[ing] (to disable warnings)

Can also use “-w” or “--w”

VMS syntax: /warn[ing] or /nowarn[ing] (to disable warnings)

Can also use “-w” or “--w”

Related options: NITPICK

XREF

Generates a cross-reference. The following sub-options are supported:

XREF=file:*filename* If “file” is used, cross-reference output will be sent to the specified file. To cancel this sub-option, use “XREF=nofile”.

XREF=merge By default, XREF prints separate function and variable cross-reference listings. If “merge” is used, XREF will merge all cross-reference output into a single listing.

To cancel this sub-option, use “XREF=nomerge”.

XREF=nfw:*number* “nfw” may be used to set the cross-reference name field width. *number* should be a value from 10 to 40. The default setting is 16.

XREF=nofunc If “nofunc” is used, XREF will omit functions from the cross-reference. To cancel this sub-option, use “XREF=func”.

XREF=novar If “novar” is used, XREF will omit variables from the cross-reference. To cancel this sub-option, use “XREF=var”.

Sub-options may be combined. For example:

lplus -xref=merge,nfw:24	under UNIX
lplus /xref=(merge,nfw:24)	under VMS

Note: XREF enables “global mode” (i.e., the GLOBAL option).

For additional information on the cross-reference, see section 6.8.

UNIX syntax: -xref or
 -xref=option,... or --xref (to cancel)

Can also use “-x” or “-x=option,...” or “--x”

VMS syntax: /xref or
 /xref=(option,...) or /noxref (to cancel)

Can also use “-x” or “--x”

Related options: GLOBAL

6



lint-PLUS Output

6.1 Overview

lint-PLUS produces the following kinds of output:

- (a) diagnostic messages
- (b) source listings
- (c) call trees
- (d) include-file trees
- (e) 'C' prototypes and Fortran shells
- (f) cross-reference tables
- (g) statistical reports

6.2. Source listing

The **LIST** option generates a source listing. If local errors are detected, the listing displays them in context. "Include" files are normally excluded; the sub-option "LIST=include" may be used to add them to the listing.

By default, long lines are clipped at column 72. Sub-options may be used to modify this behavior as follows:

Mode	Option
clip lines at a different column	LIST=clip: <i>number</i>
wrap lines at a specified column	LIST=wrap: <i>number</i>

suppress clipping and wrapping LIST=nomax

For option syntax and related information, see chapter 5.

A typical source listing is shown on the next page. Note that each line of the listing starts with three numbers: line number, levels of “include” nesting, and levels of “brace” nesting. An asterisk is added after the numeric fields to flag lines which are suppressed by “#if” or “#ifdef”.

Note: The “brace” nesting level is valid only for lines which are not suppressed by “#if” or “#ifdef”.

Sample source listing

lintPLUS rev 5.0x

primes.c

12-Jun-1998, 16:56

```

1 0 0    #include <stdio.h>
2 0 0
3 0 0    #define BASESIZE    10
4 0 0    #define FALSE      0
5 0 0    #define NTIMES     20
6 0 0    #define TRUE       1
7 0 0
8 0 0    main()
9 0 0    {
10 0 1        int count,iter,j,k,prime;
11 0 1        static char flag [BASESIZE+1];
12 0 1
13 0 1        printf ("%d iterations\n", NTIMES);
14 0 1
15 0 1        for (iter = 1; iter < NTIMES; iter++)
16 0 1        {
17 0 2            count = 0;
18 0 2            for (j = 0; j <= BASESIZE; j++) flag [j] = TRUE;
19 0 2
20 0 2            for (j = 0; j <= BASESIZE; j++)
21 0 2            {
22 0 3                if (flag [j])
23 0 3                {
24 0 4                    prime = j + j + 3;
25 0 4    #ifdef PRINT_IT
26 0 4 *                printf ("%d\n", prime);
27 0 4 *    #endif
28 0 4                    for (k = prime;
29 0 4                        k <= BASESIZE;
30 0 4                        k += prime)
31 0 4                    {
32 0 5                        flag [k] = FALSE;
33 0 5                    }
34 0 5
35 0 4                    count++;
36 0 4                }
37 0 4            }
38 0 3        }
39 0 2
40 0 1        printf ("%d primes found.\n", count);
41 0 1    }

```

6.3 Local Diagnostics

If lint-PLUS detects a “local” problem (e.g., a syntax error or questionable code), it prints the source line in question along with a short diagnostic message. An arrow is placed under the source line to show the location of the problem.

If the **VERBOSE** option is used, lint-PLUS adds background information to the diagnostic message. (Note: If an error occurs more than once in a source file, the extra text will be printed only for the first occurrence of the error.)

A typical local-error message is shown below:

```
foo = *bar+10;
      ^
main.c: line 70: Error # 29: Not a pointer.
> The operand of * or [ ] must be an array or a pointer.
```

Note: **VERBOSE** is on, for this example.

Options may be used to control or filter local diagnostics as follows:

Mode	Option
-----	-----
generate audible output	BEEP
skip repeated messages	BRIEF
enable or disable “FYI” messages	NITPICK
enable or disable warnings	WARNINGS
set maximum number of errors	MAXERROR
set maximum number of aborts	MAXFATAL

For option syntax and related information, see chapter 5.

Note: “FYI” messages are “for your information” messages which may or may not indicate problems.

6.4 Global Diagnostics

If the **GLOBAL** option is used, lint-PLUS will add “global” diagnostics to the output. Global diagnostics address a number of problems, including:

- (a) inconsistent function definitions
- (b) inconsistent variable definitions
- (c) problems with function calls

The **GLOBAL** option also produces three function tables:

- (a) list of unused functions
- (b) list of external functions
- (c) list of library functions called

Options may be used to control “global” output as follows:

Mode	Option
-----	-----
Control system-library checking	LIB
Set maximum number of errors	MAXGSE
Extra prototype checks	STRICT=proto
Extra data-type checks	STRICT=intsize
Report signed/unsigned conflicts	SIGN

For option syntax and related information, see chapter 5.

Sample “global” output is shown on the next page.

Sample global mode output

```

=====  FUNCTION DEFINITION ISSUES  =====

Function "add_ref":  type mismatch for argument #1
    declaration - file "local.c", line 50
    definition  - file "merge.c", line 1094

Function "x_write":  inconsistent return type
    prototype   - file "proto.h", line 59
    definition  - file "local.c", line 63

=====  FUNCTION CALL ISSUES  =====

Function "get_size":  inconsistent number of arguments
    call - file "local.c", line 155
    call - file "misc.c", line 289

Function "x_write":  type mismatch for argument #3
    definition - file "local.c", line 270
    call - file "main.c", line 430

=====  EXTERNAL FUNCTIONS  =====

The following functions are called but not defined:

add_dpa    add_dsa    agets      aputs      atoh
b_clr      chk_dba    dba_cpy    free_dsa

=====  UNUSED FUNCTIONS  =====

The following functions are defined but not called:

block2     get_link

=====  LIBRARY FUNCTIONS  =====

The following library functions are used:

_filbuf    _flsbuf    fclose     fflush     fgets
fopen      fprintf  fputc      fseek      ftell
getenv     isatty    printf     setbuf     setvbuf
strcat     strcmp    strcpy     strlen

=====  GLOBAL VARIABLE ISSUES  =====

Variable "decflt":  inconsistent type
    definition  - file "misc.c", line 643
    declaration - file "output.c", line 871

```

6.5 Call trees

If the **TREE** option is used, lint-PLUS will produce call trees that show the function-call structure of the source code analyzed.

For example, if the source code contains the following three functions:

```
main()          foo()          bar()
{               {               {
    foo();       abc();         bar();
    bar();       bar();         xyz();
}               }               }
```

TREE will generate the following call tree:

```
===== PRIMARY CALL TREE FOR "main" =====

*name*   - flags a recursive call
(number) - labels a duplicate subtree
(letter) - labels a continuation tree

main--+-foo--+-abc
      |      |
      |      +-bar (1)--*bar*
      |              |
      |              +-xyz
      |
      +-bar -- see (1)
```

TREE normally prints call trees to the standard output stream along with other lint-PLUS output. The sub-option “**TREE=file:filename**” may be used to send call trees to a specified output file.

By default, **TREE** clips call trees at column 72. Extra “continuation” trees are generated to display branches which are cut off. Sub-options may be used to modify this behavior as follows:

Mode	Option
clip lines at a different column	TREE=clip:number
suppress clipping	TREE=noclip

To improve readability, **TREE** normally adds extra white space to call trees. The sub-option “**TREE=nosquash**” may be used to remove the extra space.

For option syntax and related information, see chapter 5.

6.6 "Include file" trees

If the **HTREE** option is used, lint-PLUS will produce “include file” trees that show the nesting structure of the include files used by the input code.

For example, if the following source files are used:

```
foo.c:                                foo.h:
#include "foo.h"                        #include <stdio.h>
#include "bar.h"                        #include <gen.h>
#include "eval.h"                       #include "util.h"
...                                     ...

eval.h:                               gen.h:
#include "gen.h"                        #include "defs.h"
...                                   #include "flow.h"
                                   #include "globals.h"
                                   ...
```

HTREE will generate the following include-file tree:

```
----- INCLUDE-FILE TREE FOR "foo.c" -----

*name*    - flags a recursive load
(number)  - labels a duplicate subtree
(letter)  - labels a continuation tree

foo.c--foo.h--stdio.h
      |      +-gen.h (1)--defs.h
      |      |      +-flow.h
      |      |      +-globals.h
      |      +-util.h
+-bar.h
+-eval.h--gen.h -- see (1)
```

HTREE normally prints include-file trees to the standard output stream along with other lint-PLUS output. The sub-option “HTREE=file:filename” may be used to send include-file trees to a specified output file.

By default, **HTREE** clips include-file trees at column 72. Extra “continuation” trees are generated to display branches which are cut off. Sub-options may be used to modify this behavior as follows:

Mode	Option
-----	-----
clip lines at a different column	HTREE=clip: <i>number</i>
suppress clipping	HTREE=noclip

For option syntax and related information, see chapter 5.

By default, HTREE produces a separate include-file tree for every source file. The sub-option “HTREE=merge” may be used to produce a single tree for all source files.

For example:

Include-file tree without “merge”:

```

test1.c--demo.h--stdio.h
      |      +-setup.h
      +-alloc.h

test2.c--bar.h
      +-radio.h--colors.h--primary.h
      |      +-spectrum.h
      +-alloc.h

```

Same tree with “merge” enabled:

```

program--demo.h--stdio.h
      |      +-setup.h
      +-alloc.h
      +-bar.h
      +-radio.h--colors.h--primary.h
                        +-spectrum.h

```

Note: Merged trees will necessarily omit some information.

HTREE normally displays include-file names without path components. The sub-option “HTREE=path” may be used to display pathnames (where applicable).

For example:

Include-file tree without “path”:

```

test1.c--demo.h--stdio.h
      |      +-setup.h
      +-alloc.h

```

Same tree with “path” enabled:

```

test1.c--demo.h--/usr/include/stdio.h
      |      +-../misc/target/x86/setup.h
      +-alloc.h

```

For option syntax and related information, see chapter 5.

6.7 ‘C’ prototypes and Fortran shells

If an option of the form “PROTO=file:filename” is used, lint-PLUS will generate ‘C’ prototypes for functions defined by the input code. The prototypes will be stored in the specified output file.

For example, if the following source code is used:

```
double bar(x,y,p)          void foo(s,a,b)
double x;                  char *s;
double y;                  int a;
int *p;                    long b;
{                          {
    ...                    ...
}
```

PROTO will generate the following output file:

```
double bar(double,double,int *);
void foo(char *,int,long);
```

Prototypes are stored in alphabetical order.

If the sub-option “fortran” is specified, **PROTO** will generate Fortran “shells” instead of ‘C’ prototypes. (For option syntax, see chapter 5.)

Fortran “shells” are function or subroutine stubs based on the original ‘C’ routines. For example, if the ‘C’ code shown above is used, the “fortran” sub-option will generate the following output file:

```
double precision function bar(a1,a2)
double precision a1
integer a2
end

subroutine foo(a1,a2,a3)
integer a1
integer a2
integer a3
end
```

PROTO translates ‘C’ data types to Fortran data types as follows:

int, short, long, bitfield, or enum	--> integer
char	--> character
float	--> real
double or long double	--> double precision
any pointer (except function pointer)	--> integer
function pointer	--> external

Some 'C' language features are not supported (for example, variable-length argument lists). If a 'C' function uses unsupported features, lint-PLUS omits the corresponding Fortran “shell”, and adds a comment to the output file instead.

There are two special cases:

- (a) If a 'C' function name ends with one or more underscores, lint-PLUS drops the underscores from the corresponding Fortran “shell” name.
- (b) lint-PLUS represents “void” functions as Fortran subroutines.

Additional examples:

'C' function	Fortran shell
-----	-----
int foo_(a1,a2) char a1; char *a2; {}	integer function foo(a1,a2) character a1 integer a2 end
float bar(a1) int (*a1)(); {}	real function bar(a1) external a1 end
void TestIO(a1,a2) FILE *a1; double a2; {}	subroutine TestIO(a1,a2) integer a1 double precision a2 end

For option syntax and related information, see chapter 5.

6.8 Cross-reference

If the **XREF** option is used, lint-PLUS will generate a symbol-table cross-reference.

The cross-reference normally includes two tables, one for ‘C’ functions and the other for global variables. Sub-options may be used to modify the output format as follows:

Mode	Option
-----	-----
merge function and variable tables	XREF=merge
set name-field width	XREF=nfw: <i>number</i>
suppress function cross-reference	XREF=nofunc
suppress variable cross-reference	XREF=novar

XREF normally prints cross-reference tables to the standard output stream along with other lint-PLUS output. The sub-option “XREF=file:*filename*” may be used to send cross-reference tables to a specified output file.

For option syntax and related information, see chapter 5.

Sample cross-reference output is shown on the next page.

Sample cross-reference output

```

===== FUNCTION CROSS-REFERENCE =====

-D definition          -E external declaration
-P prototype          -R function pointer

static int add_ext();
    util.c      77-D      813      840

struct mac_sym *add_mac0();
    macro.c     10-P      292      323      406      453
        877      1073    1077-R      1105      1210      1800
        1952
    util.c      50-E      97

void zexit();
    setup.c      72
    sysdep.h     32-P
    util.c      146      214

===== GLOBAL VARIABLE CROSS-REFERENCE =====

-D definition          -E external declaration
-S modified variable

struct c_list CL_EXPR;
    alloc.c      266      417      933-S      960-S      1077
        1185      1219
    globals.h    82-D

char *mode_ptr;
    globals.h    364-D
    symbol.c     1819    1828-S      1840      1873      2088
    util.c      107

long xcount;
    main.c      1654-S
    globals.h    40-D
    util.c      86      112

```

6.6.9 Statistics

If the **STATISTICS** option is used, lint-PLUS will generate statistical reports.

By default, **STATISTICS** generates both local (per module) statistics and global (per run) statistics.

Local statistics cover errors:

```
Summary for "test7.c":

        6      Warning(s)
        4      Error(s)

Error      # 33 occurred 3 times.
Warning    #105 occurred 5 times.
Other messages -      2.
```

Note: The per-message breakdown is omitted in certain cases (for example, if no error occurs three or more times).

Global statistics provide function counts and I/O-related numbers:

```
Source  files processed:           40
Include files processed:          625

Source-file lines processed:       29,089
Source-file chars processed:      1,121,966

Lines processed (counting include files):  152,837
Chars processed (counting include files):  6,759,424

Global functions:                  133
Static functions:                  139
Functions (total):                 272
```

Sub-options may be used to modify the output format as follows:

Mode	Option
-----	-----
disable "global" statistics	STATISTICS=noglobal
disable "local" statistics	STATISTICS=nolocal

For option syntax and related information, see chapter 5.

7



Include Files

7.1 Overview

This chapter discusses “local” (double-quoted) include files:

#include “foo.h”

and “standard” (angle-bracketed) include files:

#include <stdio.h>

7.2 Local include files (UNIX)

UNIX lint-PLUS searches for “local” (double-quoted) include files in the following directories, using the order shown here:

- (a) Directory specified by original “#include” statement (*)
- (b) Source-file directory

- (c) Directories specified on the command line
- (d) Directories specified by environment variables (**LOCINC** or **INCLUDE**)
- (e) Current working directory

(*) If a relative path is used, the path is interpreted relative to the directory in which the current 'C' source file resides.

To specify local include file directories on the command line, use the “-I” (dash eye) option:

lplus -I../includir *.c

To specify local include file directories using the environment, set **LOCINC** or **INCLUDE** as follows:

directory-path
or **directory-path:....:directory-path**

Directory paths must be separated by colons.

7.3 Local include files (VMS)

VMS lint-PLUS searches for “local” (double-quoted) include files in the following directories, using the order shown here:

- (a) Directory specified by original “#include” statement (*)
- (b) Source-file directory
- (c) Directories specified on the command line
- (d) Directories specified by the logical **C\$INCLUDE**
- (e) Current working directory

(*) Special case: If the original path contains a forward slash, lint-PLUS replaces the slash with a colon and tries to open the modified path.

To specify local include file directories on the command line, use the “/local” option:

lplus /local=[-includir] *.c

To specify local include file directories using the environment, set **C\$INCLUDE** as follows:

directory-path
or **directory-path,...,directory-path**

Directory paths must be separated by commas.

7.4 Standard include files (UNIX)

UNIX lint-PLUS searches for “standard” (angle-bracketed) include files in the following directories, using the order shown here:

- (a) Directories specified by the **STANDARD** (-S) option
- (b) Directories specified by the environment variable **STDINC**
- (c) Host system’s “/usr/include” directory
- (d) “Local” include-file directories (see section 7.2)

To specify standard include-file directories on the command line, use the “-S” option:

lplus -S./stdinc *.c

To specify standard include-file directories using the environment, set **STDINC** as follows:

directory-path
or **directory-path:....:directory-path**

Directory paths must be separated by colons.

7.5 Standard include files (VMS)

VMS lint-PLUS searches for “standard” (angle-bracketed) include files in the following directories, using the order shown here:

- (a) Directories specified by the **STANDARD** option
- (b) Directories specified by logicals (**VAXC\$INCLUDE** or **SYSS\$LIBRARY**)
- (c) “Local” include-file directories (see section 7.3)

To specify standard include-file directories on the command line, use the “/standard” option:

lplus /standard=[-stdir] *.c

To specify standard include-file directories using logicals, set **VAXC\$INCLUDE** or **SYSS\$LIBRARY** as follows:

directory-path
or **directory-path,...,directory-path**

Directory paths must be separated by commas.

8



Magic Comments

8.1 Overview

A “magic comment” is a ‘C’ comment which may be used to modify analysis rules or settings inside a source file. Magic comments have no effect on normal program operation.

lint-PLUS supports the following magic comments:

ARGSUSED	Flags arguments as used
DISABLE	Disables local error messages
ENABLE	Enables local error messages
LINTLIBRARY	Flags a “library definitions” file
LISTING	Enables or disables a source listing
NOCHECK	Disables checking for specific function arguments
VARARGS	Flags a variable-length argument list
VOID	Marks a function result as optional

8.2 List of magic comments

ARGSUSED

If the magic comment `/*ARGSUSED*/` is placed before a function definition, it will suppress unused-argument error messages for the function.

DISABLE

May be used to disable local-error messages inside a source file. This magic comment has the following format:

```
/* DISABLE = msg#,msg#,msg#,... */
```

where the argument list specifies one or more local-error message numbers.

White space is optional. E.g., the following two comments are equivalent:

```
/*DISABLE=33,42*/
/* DISABLE = 33, 42 */
```

Note: Option switches may also be used to disable local-error messages. For additional information, see **DISABLE** in chapter 5.

ENABLE

May be used to enable local-error messages which have been disabled by **DISABLE** option switches or the **DISABLE** magic comment.

ENABLE comments have the following format:

```
/* ENABLE = msg#,msg#,msg#,... */
```

where the argument list specifies one or more local-error message numbers.

White space is optional. E.g., the following two comments are equivalent:

```
/*ENABLE=33,42*/
/* ENABLE = 33, 42 */
```

Note: Option switches may also be used to enable local-error messages. For additional information, see **ENABLE** in chapter 5.

LINTLIBRARY

If the magic comment `/*LINTLIBRARY*/` is placed at the start of a 'C' source file, lint-PLUS will treat the source file as a “library definitions” file.

A “library definitions” file may contain data-type definitions, global variable declarations, and function prototypes. lint-PLUS flags the specified variables and functions internally as “library” objects.

“Library definitions” files serve three purposes:

- (a) They allow lint-PLUS to distinguish between library functions and user functions,
- (b) They allow lint-PLUS to prototype functions in cases where the library header files don't include prototypes, and
- (c) They allow lint-PLUS to detect library interface errors whether or not the library header files are used.

Note: The lint-PLUS package includes a predefined “library definitions” file named **“syslib.c”**. If the **GLOBAL** option is used, lint-PLUS checks function calls against this file. For additional information, see section 9.5.

LISTING

Allows the user to disable or re-enable a source code listing at specific locations inside a source file.

Two commands are supported:

```
/* LISTING=OFF */    will turn the listing off
/* LISTING=ON */     will turn the listing back on
```

White space is optional. E.g., the following two comments are equivalent:

```
/*LISTING=OFF*/
/* LISTING = OFF */
```

Note: If **LIST** is not specified on the command line, `/*LISTING=...*/` has no effect. For additional information, see **LIST** in chapter 5.

NOCHECK

May be used to disable argument checking for specific function arguments.

NOCHECK comments may be placed before function definitions or function prototypes. Any of the following formats may be used:

```
/* NOCHECK # # #      ... */
/* NOCHECK #, #, #,    ... */
/* NOCHECK = # # #     ... */
/* NOCHECK = #, #, #,  ... */
```

One or more function-argument numbers should be specified. “1” stands for the leftmost argument, “2” stands for the next argument, etc. **NOCHECK** tells lint-PLUS not to check the specified arguments during “global” processing.

White space is optional. E.g., the following two comments are equivalent:

```
/*NOCHECK=2*/
/* NOCHECK = 2 */
```

NOCHECK supports functions that take up to eight arguments. To suppress argument checking for functions with longer argument lists, use **VARARGS**.

NOCHECK commands are not cumulative. If lint-PLUS encounters two or more **NOCHECK** commands for the same function, the final command will be used.

VARARGS

The magic comment `/*VARARGS*/` should be placed before every function that accepts a variable number of arguments. This magic comment suppresses inappropriate argument-list error messages.

`/*VARARGS*/` may also be placed before external function declarations and function prototypes.

Some functions accept a fixed number of arguments followed by a variable number of arguments. To flag functions of this type, use magic comments of the form `/*VARARGS#*/`, where “#” is the minimum number of arguments.

For example, `/*VARARGS3*/` may be used to flag a function which takes three fixed arguments followed by a variable number of additional arguments.

White space is optional. E.g., the following two comments are equivalent:

```
/*VARARGS3*/  
/* VARARGS 3 */
```

VOID

If the magic comment `/*VOID*/` is placed before a function definition, it will suppress “unused function result” error messages for the function.

`/*VOID*/` may also be placed before external function declarations and function prototypes.

9



General information

9.1 Overview

This chapter discusses the following issues:

- (a) Command files
- (b) Configuration files
- (c) Pointer assignments
- (d) System-library definitions
- (e) VMS CDD (Common Data Dictionary) support

9.2 Command files

Command-line arguments may be specified indirectly. If “**foo.txt**” is a text file containing option switches or filenames, the following command will add the contents of “**foo.txt**” to the lint-PLUS argument list:

lplus @foo.txt ...

Text files which are used this way are called “command files”.

Command files may specify any number of switches or filenames. There are two restrictions:

- (a) Strings inside command files must be separated by white space or newlines
- (b) Wildcards (such as “*.c”) are not supported inside command files

UNIX example:

The following commands will analyze all ‘C’ source files in the current directory (under UNIX):

```
ls *.c > foo.txt  
lplus @foo.txt
```

VMS example:

The following commands will analyze all ‘C’ source files in the current directory (under VMS):

```
dir /nodate /noheader /nosize /notrailer /output=foo.txt *.c  
lplus @foo.txt
```

Command files may be nested. For example:

If “**foo.txt**” contains the line: **-g -n -x @bar.txt**
and “**bar.txt**” contains the line: **test1.c test2.c**
then

```
lplus @foo.txt
```

will analyze the ‘C’ source files “**test1.c**” and “**test2.c**” using the option switches “**-g -n -x**”.

Command files may include comments. Comments start with a double semicolon and run to the end of a line.

For example, under UNIX, a commented command file might look like this:

```
-maxerr=50           ;; set max errors per module  
-system=host        ;; specify target system  
-tree=squash,trim    ;; set call-tree options
```

The VMS version would look like this:

```
/maxerr=50           ;; set max errors per module  
/system=host         ;; specify target system  
/tree=(squash,trim)  ;; set call-tree options
```

9.3 Configuration Files

To set command-line options automatically, create a command file named “**lplus.cfg**” and add option switches to this file. lint-PLUS will treat “**lplus.cfg**” as a configuration file.

lint-PLUS searches for “**lplus.cfg**” in the following directories:

- (a) Current working directory
- (b) Directories specified by the environment variable “**LINTCFG**” (under UNIX) or logical “**LINT\$CFG**” (under VMS)

- (c) Directories specified by the environment variable “**CSIHOM**E” (under UNIX) or logical “**IP**T\$**L**IN**T**” (under VMS)

If “**lplus.cfg**” is found, lint-PLUS adds the contents of the file to the command line.

Multiple configuration files may be used; e.g., for different projects. **LINTCFG** (or **LINT\$CFG**) should be set appropriately for users working on each project.

CSHOME and **LINTCFG** may specify one or more directories (under UNIX) using the following format:

directory-path
or **directory-path:....:directory-path**

Under UNIX, directory paths must be separated by colons.

IPT\$LINT and **LINT\$CFG** may specify one or more directories (under VMS) using the following format:

directory-path
or **directory-path,...,directory-path**

Under VMS, directory paths must be separated by commas.

9.4 Pointer assignments

lint-PLUS applies the following rules to pointer assignments:

- (a) Any pointer value may be assigned to a void-pointer variable.
- (b) Assigning a void-pointer value to a non-void pointer variable will produce a diagnostic message.

If portability checking is turned off, this will be an “FYI” message.

If portability checking is turned on, this will be a warning message.

To suppress the diagnostic message, typecast the void-pointer value appropriately.

- (c) Assigning a non-void pointer value to a different kind of non-void pointer variable will produce a warning message.

For additional information, see the ANSI C standard (X3.159-1989), section 3.2.2.3.

9.5 System-library definitions

The lint-PLUS package includes a ‘C’ source file named “**syslib.c**” which contains prototypes for standard run-time library routines. If the **GLOBAL** option is used, lint-PLUS checks function calls against this file.

“**syslib.c**” is user-modifiable. I.e., library routines supported by local compilers may be added to this file. (Note: Modifications should be bracketed by appropriate “**#ifdef**” statements.)

For additional information, see the **GLOBAL** and **LIB** options in chapter 5.

9.6 VMS CDD support

The VMS version of lint-PLUS supports CDD (Common Data Dictionary) “**#dictionary**” statements.

lint-PLUS uses the VAX ‘C’ compiler to translate “**#dictionary**” statements into the corresponding ‘C’ source code. Consequently, the compiler must be installed before “**#dictionary**” statements can be processed. Additionally, the CDD logicals must be set to the appropriate values.

If the sub-option “**LIST=include**” is used, lint-PLUS will produce a source listing that includes the translated CDD definitions. For additional information, see the **LIST** option in chapter 5.

Appendix A



Installation under UNIX

A.1 Overview

This appendix describes the procedure used to install lint-PLUS under UNIX.

Note: The lint-PLUS package includes a “license manager” daemon (**iptlmd**) which must be loaded before lint-PLUS will run. Section A.3 covers the procedure used to set up the daemon.

lint-PLUS supports several command-line options related to **iptlmd**. Additionally, **iptlmd** supports command-line options of its own. Both sets of options are listed in appendix C.

A.2 Installation procedure

1. Choose a directory for lint-PLUS. For example, the directory “**/usr/lib/lplus**” might be used.
2. Set up the lint-PLUS directory as follows:
 - (a) log in as system manager
 - (b) create the lint-PLUS directory
 - (c) go to the lint-PLUS directory
 - (d) grant users search and read rights to the directory

For example:

```
su
md /usr/lib/lplus
cd /usr/lib/lplus
chmod 755 /usr/lib/lplus
```

The license-manager daemon (**iptlmd**) should normally be started at boot time or by the system manager. However, ordinary users can start the daemon themselves, if necessary. To enable this feature, grant users write access to the lint-PLUS directory.

For example:

```
chmod a+w /usr/lib/lplus
```

3. Mount the lint-PLUS production tape and execute a command of the following form:

```
tar xvof /dev/device_name
```

Substitute the appropriate device specifier for “/dev/device_name”. For example, on a SunOS system, “/dev/rst8” might be used.

4. **tar** should load the following files from the tape:

iptlmd...	license-manager executable(s)
lplus...	lint-PLUS executable(s)
lplus.err	list of “local” error messages
lplus.hlp	lint-PLUS “help” file
syslib.c	system-library prototypes

On some systems, two or more lint-PLUS executables may be loaded. For example:

lplus_sun3	Sun-3 executable
lplus_sun4	Sun-4 executable

If two or more “lplus...” files are loaded, select the appropriate file and rename it to **lplus**.

Similarly, two or more license-manager executables may be loaded. If so, select the appropriate file and rename it to **iptlmd**.

5. Modify the user configuration for each lint-PLUS user as follows:

- (a) Set the environment variable **CSHOST** to the hostname of the system on which lint-PLUS will be installed. (Note: On most UNIX systems, the command **hostname** may be used to obtain the current host name.)
- (b) Set the environment variable **CSHOME** to the name of the lint-PLUS installation directory. A full pathname should be used.
- (c) Add the lint-PLUS installation directory to the default **PATH** directory searchlist.

For example, if `csh` (C-shell) is the local shell, add commands of the following form to the “.cshrc” file for each lint-PLUS user:

```
setenv CSIHOST localhost
setenv CSIHOME installation_directory
set PATH = ($PATH $CSIHOME)
```

Substitute the appropriate host name for *localhost*.

Substitute the appropriate directory name for *installation_directory*.

For other shells, substitute the appropriate commands.

6. lint-PLUS may now be activated. To activate lint-PLUS, use the procedure described in the next section.

A.3 Activation procedure

1. Go to the lint-PLUS installation directory (**\$CSIHOME**) and execute the following command:

```
lplus -license=activate
```

lint-PLUS should display a server code. Contact Cleanscape at (408) 978-7000 and request a lint-PLUS activation code. Cleanscape will provide you with a unique activation code based on the server code. After you obtain an activation code, execute the lint-PLUS “activate” command again and enter the activation code when lint-PLUS requests it.

2. Step 1 (the “activate” command) creates a file named **startup** in the lint-PLUS installation directory (**\$CSIHOME**). **startup** is a shell script that can be used to start (or restart) the license manager daemon. This script does not require root privileges; however, it does assume that the user has write access to the lint-PLUS installation directory (**\$CSIHOME**).

To start the daemon automatically, add a command to the system boot files which invokes “**\$CSIHOME/startup**”. To complete installation, in this case, reboot the system.

If the system boot files are not modified, users will need to start the daemon manually before running lint-PLUS.

3. If lint-PLUS is executed and **iptlmd** is not running, for any reason, lint-PLUS will print a message to this effect and terminate. If this happens, the system manager should execute “**\$CSIHOME/startup**” to start the daemon manually. Alternatively, users with write access to “**\$CSIHOME**” may start the daemon themselves.

The **iptlmd** startup process takes three minutes. If lint-PLUS is executed during this three-minute period, the program will wait for the daemon to start up before continuing.

Appendix B



Installation under VMS

B.1 Overview

This appendix describes the procedure used to install lint-PLUS under VMS.

Note: The lint-PLUS package includes a “license manager” daemon (**iptlmd**) which must be loaded before lint-PLUS will run. Section B.3 covers the procedure used to set up the daemon.

lint-PLUS supports several command-line options related to **iptlmd**. Additionally, **iptlmd** supports command-line options of its own. Both sets of options are listed in appendix C.

B.2 Installation procedure

1. Choose a directory for lint-PLUS. For example, the directory “[APPS.LPLUS]” might be used.
2. Set up the lint-PLUS directory as follows:
 - (a) log in as system manager
 - (b) create the lint-PLUS directory
 - (c) go to the lint-PLUS directory
 - (d) grant users search and read rights to the directory

The license-manager process (**iptlmd**) should normally be started at boot time or by the system manager. However, users with **SYSNAM** and **NETMBX** privileges can start the process themselves, if necessary. To enable this feature, grant users write access to the lint-PLUS directory.

3. Mount the lint-PLUS production tape and execute the following commands:

```
BACKUP/LOG device-name:LPLUS []  
DISMOUNT   device-name  
MOUNT/FOR  device-name:
```

Substitute the appropriate device name for *device-name*. For example, on an MVII system, the TK-50 device name “MUA0” might be used.

4. **BACKUP** should load the following files from the tape:

iptlmd.exe	license-manager executable
lplus.exe	lint-PLUS executable
lplus.err	list of “local” error messages
lplus.hlp	lint-PLUS “help” file
syslib.c	system-library prototypes

5. Modify the user configuration for each lint-PLUS user as follows:

- (a) Set the logical **CSHOST** equal to the node name of the system on which lint-PLUS is installed. Node names may generally be obtained using the command:

```
WRITE SYS$OUTPUT F$GETSYI (“NODENAME”)
```

- (b) Set the logical **IPT\$LINT** equal to the name of the lint-PLUS installation directory. A full path should be used.
- (c) Define a logical **LPLUS** for the lint-PLUS executable.

For example, add commands of the following form to the “**login.com**” file for each lint-PLUS user:

```
DEFINE CSHOST nodename  
DEFINE IPT$LINT [installation_directory]  
LPLUS := $IPT$LINT:LPLUS.EXE
```

Substitute the appropriate node name for *nodename*.
Substitute the appropriate directory name for *installation_directory*.

6. lint-PLUS may now be activated. To activate lint-PLUS, use the procedure described in the next section.

B.3 Activation procedure

1. Go to the lint-PLUS installation directory (**IPT\$LINT**) and execute the following command:

```
lplus /license=activate
```

lint-PLUS should display a server code. Contact Cleanscape at (408) 978-7000 and request a lint-PLUS activation code. Cleanscape will provide you with a unique activation code based on the server code. After you obtain an activation code, execute the lint-PLUS “activate” command again and enter the activation code when lint-PLUS requests it.

2. Step 1 creates a file named “**startup.com**” in the lint-PLUS installation directory (**IP\$Lint**).

“**IP\$Lint:startup.com**” is a procedure file that can be used to start (or restart) the **iptlmd** license-manager process. This procedure requires **SYSNAM** and **NETMBX** privileges. Additionally, this procedure assumes that the user has write access to the lint-PLUS installation directory.

To start **iptlmd** automatically, add a command to the system boot files which invokes “**IP\$Lint:startup.com**”. To complete installation, in this case, reboot the system.

If the system boot files are not modified, users will need to start **iptlmd** manually before running lint-PLUS.

3. If lint-PLUS is executed and **iptlmd** is not running, for any reason, lint-PLUS will print a message to this effect and terminate. If this happens, the system manager should execute “**IP\$Lint:startup.com**” to start **iptlmd** manually. Alternatively, users with **SYSNAM** / **NETMBX** privileges and write access to the “**IP\$Lint**” directory may start **iptlmd** themselves.

The **iptlmd** startup process takes three minutes. If lint-PLUS is executed during this three-minute period, the program will wait for the daemon to start up before continuing.

Appendix C



License Manager

C.1 Overview

The lint-PLUS package includes a “license manager” program (**iptlmd**) which must be started before lint-PLUS will run.

The UNIX **iptlmd** setup procedure is described in appendix A
The VMS **iptlmd** setup procedure is described in appendix B.

This appendix discusses the following issues:

- (a) activation codes
- (b) lint-PLUS options related to **iptlmd**
- (c) **iptlmd** command-line options
- (d) “reserve” files

Note: The lint-PLUS options are administrative in nature, and are not listed in chapter 5.

C.2 Activation codes

iptlmd uses a datelock mechanism to prevent the use of lint-PLUS on unlicensed systems. The datelock is built into a numeric “activation code” that allows the product to run on systems for which it is licensed.

Activation codes are assigned by Cleanscape when lint-PLUS is installed (see appendix A or appendix B). Network licensees who license more than one node will need to sign on from each node to obtain system information for activation purposes.

If significant changes are made to the host system -- for example, if a new version of the operating system is installed -- a new activation code may be required.

Appendix C - License manager

In the event that a licensed user needs to host the software on a backup system, a short-term activation code will be issued.

Purchased licenses are perpetual licenses for use of the software. Cleanscape will provide a new activation code prior to the expiration of a datelock.

C.3 lint-PLUS command-line options

The following lint-PLUS command-line options may be used to perform **iptlmd**-related operations.

Use “dash” switches (i.e., “**lplus -license=...**”) under UNIX.

Use “slash” switches (i.e., “**lplus /license=...**”) under VMS.

license=activate This option is used during the lint-PLUS installation procedure. For additional information, see appendix A (for UNIX) or appendix B (for VMS).

license=kill Terminates the license-manager process. If this option is used, **iptlmd** must be restarted before lint-PLUS can be used again. For additional information, see appendix A (for UNIX) or appendix B (for VMS).

license=mode:quit Each copy of lint-PLUS is configured for a fixed maximum number of users. If the limit is reached, lint-PLUS normally waits until a free slot opens up. “license=mode:quit” tells lint-PLUS to quit instead of waiting, when this happens.

Note: This option affects only the current lint-PLUS run. To change the mode of operation permanently, place this option in a lint-PLUS configuration file. For additional information on configuration files, see section 9.3.

license=mode:wait This option tells lint-PLUS to wait for a free user slot, if necessary. This is the default mode of operation. “license=mode:wait” may be used to reverse the effect of “license=mode:quit”.

license=report Prints a summary of current license-manager activity. The output may be redirected to a file; for additional information, see section 2.2.

license=users Prints a list of users currently running lint-PLUS. The output may be redirected to a file; for additional information, see section 2.2.

C.4 “iptlmd” command-line options

The license-manager daemon (**iptlmd**) supports several command-line options of its own. The **iptlmd** options are listed below.

To use these options under UNIX, locate the **iptlmd** startup script (normally **\$CSIHOMe/startup**) and modify the script appropriately.

To use these options under VMS, locate the **iptlmd** startup script (normally **IPT\$LINT:startup.com**) and modify the script appropriately.

Note: The switches shown here expect to see a space between the “dash letter” part and the argument list (if any).

-e *dir:....:dir* The lint-PLUS installation procedure adds this switch to startup command(s) automatically.

The argument list specifies directories which contain license keys. Full directory names must be used; i.e., relative paths are prohibited. If more than one directory is specified, directory names must be separated by colons.

-l *logfile* (Lower-case ell.) Specifies a license-manager log-file name. *logfile* must be a full directory name; i.e., relative paths are prohibited.

-m *size* Specifies the maximum license-manager log file size. The *size* parameter may be expressed as a number of kilobytes (###k) or a number of megabytes (###m). Decimal points are allowed. Examples:

```
iptlmd -m 100k ...      100 kilobytes
iptlmd -m 0.5m ...      one-half megabyte
```

If the log file grows past the specified limit, **iptlmd** will copy it to “**logfile.old**” and start a new log.

-r *filename* Specifies a “reserve” file. “Reserve” files may be used to reserve licenses for specific users or hosts. For additional information, see section C.5.

-v *number* Controls log-file “verbosity”. The default value is three. Lower values of *number* will produce less output, and higher values will produce more output.

C.5 “Reserve” files

Licenses may be reserved for particular users or hosts. I.e., if this feature is used, specified users or hosts will be guaranteed a specified number of licenses, regardless of the number of users running lint-PLUS at any given time.

To use this feature, create a “reserve” file. The “reserve” file is a text file containing one or more lines in the following format:

lplus:*set:client,...,client:number*

Each *client* entry should be a username (e.g., “wendy”) or a hostname with a prepended “at” sign (e.g., “@gumby”).

set should be a unique name for the group of users or hosts specified by the *client* entries on the current line. For example, for developers working on a specific project, the project name might be used.

number specifies the number of licenses to be reserved for the current group. (Note: The total number of licenses allocated by the “reserve” file can’t exceed the number of licenses which have been purchased.)

After the “reserve” file is created, use the “-r” switch described in the previous section to load the file.

Examples:

```
# Note: "reserve" files may have comment lines inter-
# spersed with command lines. Comment lines start
# with "#".
```

```
# Reserve three slots for Wendy, Jeff and Doug.
lplus:devteam:wendy,jeff,doug:3
```

```
# Reserve one slot for the "gumby" system.
lplus:lab:@gumby:1
```