sc_util

COLLABORATORS

	TITLE : sc_util	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

sc_util

1.1 sc_util.guide

```
cpr         CodePRobe - Source level debugger
cprk        CodePRobe - Cross-debugger kernel
cprx        CodePRobe - Cross-debugger version
cover       Analyzes coverage data
demangle    Converts C names into C++ name
diff        Determines the differences between two files
grep        Searches for and prints regular expressions
gst         Manages global symbol tables
guiprof     Graphical code profiler
hypergst    Displays the contents of GSTs
lctosc      Converts older V5 options to newer options
lprof       Generates run-time statistics
lstat       Analyzes and prints run-time statistics
mkmk        Creates makefiles
omd         Disassembles object modules
oml         Manages libraries
sc          SAS/C Development System driver
sc5         SAS/C Development System V5 compatible driver
scmsg       Searches for and corrects errors and warnings
scompare    Generates patch files
scopts      Sets compiler options
scsetup     Sets up a new project
slink       SAS/C Development System Linker
smake       Maintains and updates records of file dependencies
smfind      Finds strings in text files
spatch      Applies patches to files
splat       Searches and replaces patterns matching regular expressions
sprof       Determines the time a program spends in each function
tb          Displays traceback information
```

1.2 cpr

```
cpr - CodePRobe - Source level debugger

Synopsis
```

```
   cpr [ options ] program [program parameters]

Description

   CodePRobe is a powerful source-level debugger that enables you to
   monitor, line-by-line, the behavior of your C program, Assembler
   program, or both.

See Also

    cpr commands  ,  cpr options
```

1.3 cpr options

```
-buffer size

   Specifies the size of the Dialog window buffer.  By default, the Dialog
   window saves the last 4096 bytes displayed so that you can scroll
   backwards and review output.  "size" is the new buffer size in bytes.

-cli

   instructs CodeProbe to invoke your application as a Shell (CLI)
   process.  CodeProbe passes arguments to the application through
   the normal command line interface.  This option is the default if
   CodeProbe is invoked from a Shell. The option is necessary if
   CodeProbe is invoked from the Workbench screen, and you want your
   program to run as if invoked from a Shell.

-command commands

   executes the specified debugger commands at startup.  The
   commands are executed after go main if the -startup option is not
   specified or after the profile script if -startup is specified.
   For example, the command

     cpr -command "proceed; display fahr" program-name

   executes to main, steps over 1 line of code, and displays the
   variable fahr before giving control to you.

-i

   sets up a screen in interlace mode. By default, CodeProbe opens a
   new screen using the specifications set up by Preferences for
   Workbench screens. To force a screen to be opened in interlace
   mode, include the -i option before typing the application command
   name.

-line

   starts CodeProbe in line mode. See "Running CodeProbe in Line
   Mode," earlier in this chapter, for more information.
```

-nommu

tells CodeProbe not to use the MMU. This option is useful only if you have a 68020 or higher CPU.

If your machine is running Enforcer and has a 68020 or higher CPU, CodeProbe tries to use the MMU to stop references to illegal memory. For these machines, specifying -nommu tells the debugger not to verify memory in the MMU tables before trying to access the memory. However, if Enforcer is running, you will still not be able to display illegal memory.

-noprofile

suppresses execution of the CodeProbe startup file. When CodeProbe is invoked, it automatically executes a script file named cprinit, which must be located in either the current directory or ENV:sc. If the startup file is in the current directory, CodeProbe does not search ENV:sc.

-screen screen-name

tells CodeProbe to open its windows on the named public screen. By default, CodeProbe creates a new public screen named SC_CPR.1 when it is invoked. If you specify -screen, CodeProbe looks for an existing public screen with the specified name, and if it finds one, CodeProbe uses that screen to display its windows. If a screen with the given name does not exist, CodeProbe creates a new one with that name. To use the Workbench screen, specify -screen workbench.

Public screens are a feature of Intuition 2.0. For users of earlier versions of Intuition, this option is not generally useful. However, specifying -screen workbench causes CodeProbe to use the Workbench screen on older releases.

-startup

suppresses the automatic go main that is normally executed by the debugger on startup. This option is useful you want to step through the startup code or debug constructors or autoinitialization functions that run before main. If you specify -startup, the application process does not perform any type of initialization before control is given to you.

If the quit, start, or restart commands are invoked and an application process has not exited, the debugger normally calls exit to clean up any process resources that may not have been freed. However, if you invoked CodeProbe with the -startup option, exit is not called.

-w

runs CodeProbe on the Workbench screen. Specifying -w is equivalent to specifying -screen workbench.

-wb

tells CodeProbe to invoke the application as a Workbench process.

```
   It passes arguments to the application through a Workbench
   startup message. This option is the default if CodeProbe is
   invoked from the Debug icon. The option is necessary if CodeProbe
   is invoked from a Shell, but you want the application to run as
   if invoked from the Workbench screen.

-wdialog left top width height
-wregister left top width height
-wsource left top width height
-wwatch left top width height

   specifies startup window coordinates for the Dialog, Register,
   Source, and Watch windows, respectively.  Window coordinates are
   measured in character positions.  A width or height of 0 causes
   the window to extend to the screen border on the right or bottom,
   respectively.  For example, to open the Source window in position
   (0,1) and make the Source window 50 characters (columns) wide and
   12 lines long, you would specify:

       cpr -ws 0 1 50 12 program-name

   NOTE: The Register and Watch windows are not displayed on
   startup.  However, when they are opened by pressing the
   appropriate function key (either F1 or F4), they will open
   to the coordinates specified on the command line.

-.
   suppresses the copyright banner that is displayed at startup.
```

1.4 cprk

```
cprk - CodePRobe - Cross-debugger kernel

Synopsis

   cprk [option]

Description

   This kernel program is a version of CodePRobe, a powerful source-level
   debugger that enables you to monitor, line-by-line, the behavior of
   your C program, Assembler program, or both.

   This version is to be run on a machine which will be debugged remotely,
   via the  cprx  program.

   cprk supports the following communication parameters:

-pipe name

   specifies that a named pipe is to be used instead of the serial
   port.  If name is not specified, pipe:cpr is used.  CPRX or CPRK
   will open two pipes using name as a base, name_d and name_k.

   To communicate successfully over a network, you must invoke
```

```
   either CPRX or CPRK with a pipe filename that refers to a pipe on
   the other machine.  If your network allows you to refer to a pipe
   device on an attached machine by specifying

   net:pipe/pipename

   then you can start CPRK on the target machine using

    cprk -pipe net:pipe/cpr

   and start CPRX on the host machine with the -pipe option without
   a pipename.

-device  name

   specifies that the named device should be opened for
   communications if -pipe was not specified.  The default is
   serial.device.

-unit number

   specifies the unit number of the communications device if -pipe
   was not specified.  The default is zero (0).

-speed  number

   specifies the baud rate of the communications device if -pipe was
   not specified.  The default is the value is set with the Serial
   editor in the Prefs drawer from Workbench.

See Also

    cprx
```

1.5 cprx

```
cprx - CodePRobe - Cross-debugger version

Synopsis

   cprx [ options ] program [program parameters]

Description

   This is the Cross-Debugger version of CodePRobe, a powerful
   source-level debugger that enables you to monitor, line-by-line, the
   behavior of your C program, Assembler program, or both.

   This version of cpr allows you to control a program running under the
    cprk  kernel program on another machine, via a serial or NetWork link.

See Also

    cprx commands  ,  cprk  ,  cprx options
```

1.6 cprx commands

In addition to all regular cpr commands , cprx also recognises the following:

finish

The program being debugged is automatically terminated if it has not yet completed execution.

quit

CPRK continues running, so you can start another CPRX session. The program being debugged is automatically terminated if it has not yet completed execution.

1.7 cprx options

In addition to all regular cpr options , cprx also recognises the following:

-pipe name

specifies that a named pipe is to be used instead of the serial port. If name is not specified, pipe:cpr is used. CPRX or CPRK will open two pipes using name as a base, name_d and name_k.

To communicate successfully over a network, you must invoke either CPRX or CPRK with a pipe filename that refers to a pipe on the other machine. If your network allows you to refer to a pipe device on an attached machine by specifying

net:pipe/pipename

then you can start CPRK on the target machine using

cprk -pipe net:pipe/cpr

and start CPRX on the host machine with the -pipe option without a pipename.

-device name

specifies that the named device should be opened for communications if -pipe was not specified. The default is serial.device.

-unit number

specifies the unit number of the communications device if -pipe was not specified. The default is zero (0).

-speed number

specifies the baud rate of the communications device if -pipe was not specified. The default is the value is set with the Serial editor in the Prefs drawer from Workbench.

-symfile filename

specifies the location of the executable file on the host machine. If the executable file is located in different places on the target and host, then use this option, to specify where the executable file resides on the host machine. CPRK still looks in the location specified by target-executable-filename.

-x

strips all debugging information from the host's version of the executable file, copies the stripped version to the target machine, and tells CPRK to invoke the just-transmitted version. This option allows CPRK to run constantly without changing disks or rebooting (as long as your program does not crash).

1.8 cover

cover - Analyzes coverage data

Synopsis

cover [options] [datafile]. . .

Description

The cover utility analyzes the coverage data produced when you compile your program with the COVERAGE option, link with the object file covutil.o, and run your program. Using this data, you can determine which lines of code in your program were executed and which were not. This information can help you design test cases that exercise all paths through your program.

See Also

cover options

1.9 cover options

dir=directory-pathname

specifies the directory path or paths to search to locate the C source files.

merge filename

tells cover to read the specified data files, merge the information contained in them, and write the merged data to filename.

```
nosource

   tells cover not to read in the C source files.
```

1.10 demangle

```
demangle - Converts C names into C++ names

Synopsis

   demangle C-name C-name...

Description

   The demangle utility converts the specified C names that were
   generated by the C++ translator into their original C++ forms.

   If you run sc with the cxxonly option, the resulting ..c file
   contains C names that were generated by the translator.  To read
   this file, you may need to use the demangle utility to convert
   the C names.

   If you compiled with the debug option, the debugging information
   in the object module refers to the names generated by the
   translator.  In many cases, the debugger can also display the C++
   name.  If you want to run the debugger from an AREXX script, you
   may need to use demangle to convert the C names.

Example

     > demangle bar__3foo

       foo::bar


   bar__3foo is the mangled C name as it would appear in the
   generated ..c file.  foo::bar is the demangled C++ name as it
   would appear in your C++ source file.
```

1.11 diff

```
diff - Determines the differences between two files

Synopsis

   diff [>destination] [ options ] file1 file2

Description

   The diff utility determines the differences in contents between two
   files and describes the lines that you should delete from, add to, or
```

change in file2 to make it look like file1. To help you find those lines, diff identifies the line numbers and gives the text of the lines.

If you enter the diff command without specifying filenames, diff displays the version that you are using and describes the options available.

Output from this utility is sent to the standard output device, unless you redirect it by specifying the -o option or by entering a greater than (>) sign followed by a destination.

NOTE: Appendix 1, "diff File-Matching Algorithm," discusses the theory of file matching and how it affects diff.

See Also

diff error messages , diff options

1.12 diff options

-bnn

sets the size of the I/O buffer to the value specified by nn. The default I/O buffer size is 4K, and the maximum size is limited by the amount of available system memory.

-c

displays only those lines common to both files.

-Fnn

sets the column number where you want diff to begin comparing lines. For example, if you set nn to 25, diff ignores the first 25 characters on each line.

-Lnn

sets the column number where you want diff to stop comparing lines. For example, if you set nn to 75, diff ignores the characters beyond column 75 on each line.

-lnn

defines the number of lines, nn, per file that can be handled by diff. diff uses two tables (one for each file) to keep track of the lines read in from each file. The default size of each of these tables is 2000; the maximum number is limited by memory availability.

-ofile

sends the output to the specified file (or device). You can use this option instead of the AmigaDOS redirection command. You can send the output to a different device by specifying the device name. For

example, to send output to the printer, specify -oprt:. To send output to a file on drive df0:, specify -odf0:filename.

-p

filters out unprintable characters from the input stream. You can use this option to clean out an AmigaDOS binary file.

-q

suppresses any messages if there are no differences between the specified files.

-w

ignores differences related solely to space or tab characters and reduces sequences of these characters to a single space. This option also removes trailing blanks. When you specify the -w option, diff uses additional memory to create a third table for each line in its compressed form.

1.13 diff error messages

Can't open file

indicates that diff cannot open the named file. The file may not exist or it may be protected.

Diff I/O Error

indicates an I/O problem.

Improper -l specification: nn

indicates that diff cannot interpret the nn value specified with the -l option as a number.

Internal Error: ...

indicates an internal error. Please contact the Technical Support Division at SAS Institute if you see this error message. The Technical Support representative will need the following information:

> the version of diff you are using

> the exact wording of the command line you used

> the exact wording of the message you received

> the size of the files you were trying to compare.

Line table overflow

indicates that one of the two line tables, in which diff stores the lines from each file while they are being compared, is too small to

hold the lines in one of the files. Try increasing the line table size with the -l option.

Not enough memory for nn lines per file

indicates that you have used the -l option to increase the line table size, but you do not have enough memory to increase it by the amount you specified.

Out of memory

indicates that diff is out of memory. Try decreasing the line table size with the -l option if possible. Do not use the -w option. You can also try decreasing the size of the I/O buffer by using the -b option; however, this action makes diff run slower because more disk accesses may be required. If you are using a hard disk, this factor may be negligible.

Too many file names: ...

indicates that you have made an error on the command line in such a way that diff assumes you are attempting to operate on more than two files.

Unrecognized option

indicates that you specified an option that diff does not recognize.

1.14 grep

grep - Searches for and prints regular expressions

Synopsis

grep [>destination] [options] pattern file . . .

Description

The grep utility searches the files you specify for all the lines that contain the specified pattern. For each file in which it finds the pattern, grep displays, on the screen, the filename, line number, and contents of the line that contains the matching string. The patterns that you can specify can be specific or general, and they are sometimes referred to as regular expressions. In this text, they are referred to as patterns.

To specify the files that you want grep to search, you can use the AmigaDOS wildcards (eg. #?).

Specifying the Pattern

The following list describes the special characters that you can use to specify the pattern for which you are looking:

.

is the grep wildcard character. A period will match any single character, including a space, tab, newline, or control character.

A period followed by an asterisk (*) tells grep that any number of any characters can be present. The .* sequence is equivalent to the #? sequence for AmigaDOS commands.

""

are used to enclose patterns that include space characters.

[ ]

are used to search for any one character from a set of characters by enclosing that set of characters inside square brackets. A set of characters enclosed in square brackets is called a character class. A character class will match one single character. For example, to search for lines that contain vowels, you can enter [aeiou].

n(--) -

is used to specify a range. Inside a character class, you can specify a range of characters by entering the first character in the range and the last character in the range separated by a hyphen. The first character must occur alphabetically before the second. For example, to find any one of the lowercase alphabetic characters, you enter [a-z].

!

is used as the negation character when included as the first character in a character class. An exclamation point tells grep to find any character that is not in the character class. For example, to find any one character that is not a lowercase alphabetic character, enter [!a-z]. To find any one character that is not a lowercase alphabetic character and is not the end-of-line character, enter [!a-z\N].

*

tells grep that the preceding character does not have to be present in the pattern but can be present an unlimited number of times. In other words, an asterisk means "zero or more occurances of."

A period followed by an asterisk tells grep that any number of any characters can be present. The .* sequence is equivalent to the #? sequence for AmigaDOS commands.

NOTE: Following AmigaDOS conventions, if the * is used inside quotes, it is treated as an escape character, so will need to be doubled to actually pass a * to grep. In other words, if the search string is "foo* bar", you will need to type "foo** bar" instead.

+

tells grep that the preceding character must be present in the pattern at least one time but can be present an unlimited number of times. In other words, a plus sign means "one or more of." For example, to find all lines in your file that contain one or more of the lowercase letter

t, enter t+.

^

tells grep that the pattern must begin in column 0. Enter the caret (^) at the beginning of the pattern. grep will look only for those lines in which the pattern occurs beginning in column 0. For example, to find all lines that begin with a left parenthesis, enter ^(.

$

tells grep that the pattern must occur at the end of a line. Enter the dollar sign at the end of the pattern. For example, to find all lines that end with a right parenthesis, enter)$.

\

is the escape character. If you want to search for any of the special characters included in this list, you must enter a backslash before that character. For example, to find all lines that contain a caret, enter \^. Similarly, if you want to search for a backslash, enter two backslashes, \.

You also need to use the backslash to identify certain additional characters. These characters are as follows:

\b backspace
\n newline
\r carriage return
\s space
\t tab
\xij control character identified by hexadecimal digits ij

For example, to locate all lines containing the Control-g character, enter \x07.

See Also

grep error messages , grep options

1.15 grep options

-c

prints the total number of matched lines.

-f

prints only the names of all files in which grep finds a string that matches the pattern.

-n

tells grep not to display line numbers.

-p

displays only printable ASCII characters. Non-printable characters are filtered out. This option is useful if you are searching a binary file that may contain control characters.

-q

does not display filenames or line numbers.

-s

displays the names of all files that grep searches. Normally, grep displays only those filenames in which it found a match for the pattern.

-v

prints only the lines in which a match of the pattern is not found.

-V

prints the version number of grep.

-$

tells grep not to distinguish between upper- and lowercase. For example, if you use this option, the pattern int would match int, INT, Int, and INt.

1.16 grep error messages

Bad character class

indicates that grep is unable to interpret a character class in the pattern you have asked it to find. Check to see if you have used any special symbols that you did not intend to use. This message also can be generated if, in specifying a character class and using the minus (-) character to indicate a range of characters, the first character in the range does not alphabetically precede the second (for example, [d-a]).

Bad pattern

indicates that grep is unable to interpret the pattern you are asking it to find. You may have used a special character in the pattern that does not make sense in this context. Check to see if you have forgotten to escape a special character.

Can't find file(s) ...

indicates that grep is unable to find one or more of the files you have asked it to search. The files may not exist, or you may have misspelled the filenames.

Can't open ...

indicates that grep is unable to open the named file. The file may not exist or may be protected.

Closing] not found

indicates that grep believes that you have asked it to search for a pattern that contains a character class, but you have omitted the right bracket (]) that terminates the character class. If not, you may have included a left bracket ([) in the pattern but forgot to use a backslash (\) before it.

Empty character class

indicates that a character class you have used in your pattern has no members, for example [ ].

Improper hex specification

indicates that you have used the x escape sequence but have not followed it with two recognizable hexadecimal digits.

Incompatible combination of options

indicates that the options with which you have invoked grep are contradictory.

Internal Error: ...

indicates an internal error. Please contact the Technical Support Division at SAS Institute if you see this message. The Technical Support representative will need the following information:

> the version number of grep you are using

> the exact wording of the command line you used

> the exact wording of the resulting error message.

Invalid option

indicates that you have used a command line option that grep does not recognize. This message also can be generated if your pattern begins with a minus sign (-) but you have not escaped it or enclosed the pattern in double quote marks (").

No beginning double quote in pattern

indicates that grep has detected double quote marks (") in a pattern that did not start with a double quote. You may need a backslash (\) before the double quote.

No file arguments provided

indicates that grep believes you have specified a pattern but no

filenames. This message also may be generated if you invoke grep with a filename but no pattern.

No pattern or file arguments given

indicates that grep cannot find either a pattern or filename on its command line.

Out of space

indicates that grep has run out of space in constructing its pattern representation. Please contact the Technical Support Division at SAS Institute if you see this message.

Pattern ill-formed: no terminating double quote

indicates that you started the pattern with a double quote (") but failed to terminate it with one.

Too few arguments to GREP

indicates that grep demands at least two arguments on its command line: a pattern and at least one filename.

1.17 gst

gst - Manages global symbol tables

Synopsis

```
gst [gst-filename] [ options ] [symbol-name . . . ]
```

Description

A Global Symbol Table (GST) is a collection of all the information defined in a set of header files and stored in a format that the compiler can use easily and quickly.

For information on creating and using GSTs, refer to Chapter 4, "Using the System Header Files to Access Libraries," in SAS/C Development System Library Reference.

The gst utility helps you manage GSTs and extract information from a GST.

To load a GST into memory, specify the GST command as follows:

```
gst gst-filename
```

The gst utility loads the GST into memory and then exits.

NOTE: You should do this only if you plan to remove the disk containing the GST from the drive.

To display information about a symbol contained in a GST, specify the

```
   gst command as follows:

  gst gst-filename symbol-name

   Some symbols may occur more than once in the GST.  For example, a
   symbol may occur once as a structure tag and again as an identifier.
   The gst utility prints all definitions of the same symbol.

See Also

    gst error messages ,  gst options
```

1.18 gst options

```
list

   lists all GSTs currently in memory.  You can use this option to
   determine which GSTs to unload if you need more memory.

[name=]filename

   specifies a GST file created using the makegst option in the sc
   command.  You do not need to specify name= unless you want to load a
   GST that has the same name as a keyword (for example, name=symbol).
   If you use the special name anygst, the gst utility uses the
   first GST on its list, as reported by gst list.

[symbol=]symbol-name

   displays the type of symbol name and the filename and line number where
   symbol-name is defined.  The type may be one of the following:
   n(Preprocessor) tag struct, union, or enum name identifier variable or
   function name typedef typedef include #include filename preprocessor
   preprocessor macro or #define

   You do not need to specify symbol= unless you want information about a
   symbol that has the same name as a keyword (name, unload, and so on).

unload

   forces a GST to be unloaded from memory.  If you also specify symbol
   names on the command line, gst prints all definitions for those symbols
   before it unloads the GST.

verbose

   prints the full definition of symbol-name.
```

1.19 gst error messages

```
No GST file specified
```

indicates that you did not specify a GST filename.

Can't find GST file "filename"

indicates that gst cannot find the file specified.

Symbol "symbol-name" not found

indicates that gst could not find the specified symbol.

Not enough memory to load GST

indicates that the GST file could not be loaded due to insufficient memory.

1.20 hypergst

hypergst - Displays the contents of GSTs

Synopsis

```
hypergst [gst-filename]
```

Description

The hypergst utility allows you to browse the definitions of symbols and data structures in GSTs (Global Symbol Tables) that are loaded into memory.

hypergst works with the AmigaGuide hypertext system. For information on using AmigaGuide, see Chapter 2, "Getting Help," in SAS/C Development System User's Guide, Volume I: Introduction, Editor, Compiler.

To browse the definitions in a GST, the GST must be loaded in memory. You can invoke the hypergst utility and load a GST in one of two ways:

> You can enter the hypergst command followed by the name of the GST file that you want to browse.

> From the Workbench, you can click on the HyperGST icon, then hold down the Shift key and double-click on the icon for the GST file that you want to browse.

If the GST that you want to browse is already loaded into memory:

> You can enter the hypergst command without specifying a GST filename. hypergst displays a screen listing the names of all GSTs in memory. To select a specific GST, click on the name of the GST file.

> You can double-click on the HyperGST icon. hypergst displays a screen listing the names of all GSTs in memory. To select a specific GST, click on the name of the GST file.

After the GST is loaded, hypergst displays a screen containing buttons for the various kinds of symbols defined in the GST. These buttons are as follows:

> Data Items
> Prototypes
> Typedefs
> Structs/Unions/Enums
> Preprocessor Symbols

If you click on one of these buttons, hypergst displays an alphabetical list of all symbols of that type in the GST. If you click on the name of a symbol, hypergst displays a symbol information screen for that symbol. This screen may contain other buttons that reference header files, structure tags, type definitions, and so on. You can click on any of these buttons to display more information about the highlighted item.

See Also

hypergst error messages

1.21 hypergst error messages

No GST files in memory

indicates that you have not loaded any GST files into memory. For example, you see this message if you invoke hypergst immediately following a reboot and do not specify any arguments.

1.22 lctosc

lctosc - Converts older V5 options to newer options

Synopsis

lctosc [>file] [lc options]

Description

This utility will convert V5 'LC' options to the newer 'SC' option style. It prints the converted options to stdout. To keep and use these converted options, you will need to redirect the output of lctosc to a file using the AmigaDOS '>' output redirection operator. The new driver looks for options in a file named "SCOptions".

You may either type the options to be converted on the command line, or redirect the input to lctosc from a file using the AmigaDOS '<' input redirection operator. The old driver, LC, looked for options in a file named "SASCOPTS".

1.23 lprof

lprof - Generates run-time statistics

Synopsis

```
lprof [-t=n] [>prog-output-dest] program [program-options]
```

Description

The lprof utility generates run-time statistics for a program. lprof records the amount of time spent in each routine that is called as your program runs.

To use lprof with your program, you must compile your program with the debug option.

After lprof loads your program, it runs the program and gathers statistics by examining the current program counter at given intervals. lprof writes your program's statistics to the file named prof.out in the current directory. You can use the lstat utility to produce a report from those statistics.

lprof only supports one option:

-t=n

specify the interval, in milliseconds, between lprof statistics checks on your program.

See Also

lstat

1.24 lstat

lstat - Analyzes and prints run-time statistics

Synopsis

```
lstat [>destination] [ options ] program [profile]
```

Description

The lstat utility analyzes the profile statistics created by lprof .

The profile is the name of the output file created by lprof. lstat first looks in your current directory for the default file produced by lprof, prof.out. If this file is not in your current directory, or if you have changed its name, you must specify its full pathname as the value for profile.

You must specify the program name. lstat uses this name to get debugging and symbol information for its report.

```
   NOTE: This utility assumes that you have linked the program with
   the addsym linker option.  The addsym linker option allows
   the profiler to produce statistics for library functions.
   If you link your program using the link and debug options
   in the sc command, the linker automatically links your
   program with the addsym option.  If you are linking with a
   separate slink command, you must specify the addsym linker
   option.


   If the report produced by lstat shows that a large percentage of time
   is spent in one program module, you may want to

      > redesign the module

      > incorporate the module into the calling routines, thus eliminating
  calling overhead

      > rewrite the routine in assembly language

      > restructure your program.

   You also may find that most of the work being done involves the routine
   in question, and the percentage of time taken is reasonable.

See Also

    options
```

1.25 lstat options

```
-z

   tells lstat to display statistics for all subroutines even if they were
   not encountered in profiling.  By default, lstat does not report
   subroutines that it does not encounter.

-f

   tells lstat to display full statistics for each subroutine, indicating
   the line numbers within a module that it encountered.  By default,
   lstat displays only summary information about the subroutine.

-t=n

   tells lstat to display only those routines that have at least n hits.
   By default, lstat prints all subroutines that it encountered even once.
```

1.26 mkmk

mkmk - Creates makefiles

Synopsis

mkmk [options] [filename...]

Description

The mkmk utility creates makefiles. mkmk examines the source and header files you specify, determines which files depend on which other files, and generates a makefile. If you do not specify any filenames, mkmk examines all the files in the current directory ending with .c, .cxx, .cpp, or .cc.

mkmk supports the following options:

target=filename

specifies the name to use in the makefile for the target executable module. If not specified, the root part of the name of the first .c, .cxx, .cpp, or .cc file in the file list is used.

force

tells mkmk to overwrite an existing makefile.

makefile=filename

tells mkmk to write the makefile to the specified filename. The default filename is smakefile.

After running mkmk, you can build your project by entering smake from the Shell or by double-clicking the Build icon from the Workbench. If you add any new files to your project, or if you add or delete any #include directives in existing files, you need to add them to the generated makefile either by editing the makefile or by rerunning mkmk.

The source code for mkmk is in the sc:extras/mkmk directory.

1.27 omd

omd - Disassembles object modules

Synopsis

omd [>destination] [-x] object [source]

Description

The omd (Object Module Disassembler) utility program disassembles an

```
   object file produced by the SAS/C Compiler and produces a listing
   consisting of assembly language statements (interspersed with the
   original C source code if you specified a source filename).

   object is the object filename.  You must specify the complete filename,
   including the .o extension.

   source is the source filename.  If you specify source, you must specify
   the complete source filename, and the source file should have been
   compiled with the debug compiler option.  The debug option allows omd
   to associate source lines with the object code they generated. If you
   did not use the debug option, C source lines will not appear in the
   output produced by omd.

   omd only supports one option:

  -x

      This option sets the size of the buffer used to hold the
      external symbol section of the object module.  For example,
      -x250 establishes a buffer that can hold 250 external symbols.
      The default size is 200.  You should increase the buffer size
      only if omd reports that there are too many external symbols.
```

1.28 oml

```
oml - Manages libraries

Synopsis

   oml [<cmdfile] [>listfile] [ options ] libfile [command [module. . . ]]. .

Description

   You can use the oml (Object Module Librarian) to manage your libraries.
   oml allows you to:

      > create library files
      > list the modules in a library file
      > delete modules from a library file
      > replace old modules with new modules.

   A link library is a group of object modules, each of which was
   originally in a separate file and consisted of one or more subroutines.
   Some advantages of using a link library are as follows:

      > The subroutines in a library can be used by several programs.
      > When you are linking your program, you specify only the library
  file instead of several individual object modules.
      > The linker includes only those modules in the library that are
  required by your program.

   Each module within the library is identified by a module name, which is
   placed in the object module by the program (the SAS/C Assembler or
   Compiler) that generates the module.  The assembler and compiler use
```

the name of the object file. If a module does not contain a module name, oml assigns a module name of the form $nnn, where nnn is a decimal number.

A module may define one or more public symbols. A public symbol is something in the module that is available to other modules, such as global variables or functions. When the linker resolves external references for a program, it examines each module in each library you specify. It decides which of the modules are required by the program it is linking by looking at the module's list of public symbols. If any of the program's unresolved external references match any of the module's public symbols, that module will be included in the executable module.

When writing the modules for your library, be careful to avoid duplicate names for public symbols. If you create a library that defines a symbol more than once, oml issues a warning message. If you then link with that library without correcting the problem, the linker may include the wrong module or give a duplicate symbol error.

To invoke oml, enter the oml command followed by a library name, as follows:

```
oml mylib.lib
```

oml waits for you to enter commands (and module names, if necessary) from the keyboard. oml executes the commands as you enter them and displays a list of any duplicate symbol names that it finds. After you have entered all your commands, enter a Control-backslash (Control-\) to terminate oml. You also can include the commands and module names on the oml command line. If you enter commands on the command line, oml executes those commands and terminates. For example, the following command replaces the module ftoc.o in mylib.lib with the version of the same module in your current directory:

```
oml mylib.lib r ftoc.o
```

oml replaces the module, lists any duplicate symbol names it finds, and terminates. You also can enter some commands on the command line followed by some from the keyboard. If you include an at (@) sign at the end of the command line, oml executes the commands you enter on the command line and then waits for you to enter additional commands from the keyboard. You must enter a Control-\ to terminate oml. For example, the following command replaces the module ftoc.o in mylib.lib with the version of the same module in your current directory and then waits for you to enter additional commands from the keyboard:

```
oml mylib.lib r ftoc.o @
```

You also can enter the commands into a file and tell oml to use that file as input. You can tell oml to use that file by either of the following methods:

> entering a less than (<) sign followed by the filename (<cmdfile) as the second item on the oml command line
> entering an at (@) sign followed by the filename (@cmdfile) as the last item on the oml command line.

```
   oml sends its output to the screen unless you redirect it by specifying
   a greater than (>) sign followed by a filename.

   oml produces warning messages if either of the following are true:

      > A module that you want to delete or extract is not in the library.
      > Any module in the library includes a second definition for a
  public symbol.

See Also

    oml commands  ,  oml options
```

1.29 oml commands

```
r file file ...

   replaces the named object files in the library or adds them to the
   library, if they are not already present.  Each module should be in a
   separate object file.  To replace modules within a library, make sure
   that the module contains a module name and that the filename is the
   same as the module name.

d module module ...

   deletes the named modules from the library.  oml assigns module names
   to those modules without names; therefore, you may need to get a
   listing of names in the library (using the l command) to determine the
   name of the module that you want to delete.

x module module ...

   extracts the named modules from the library and creates separate files
   using the same names.  You may need to get a listing of names in the
   library (using the l command) to determine the correct name of the
   module that you want to extract.  If the module name contains a
   pathname, oml attempts to create a file with that name.  If the module
   name does not contain a pathname, oml creates the file in the current
   directory unless you specify a different pathname with the -o option.
   You can use an asterisk (*) to specify that you want all modules
   extracted. oml terminates if it cannot extract a module.  You cannot
   extract and replace the same module with the same oml command line.

l

   generates a listing of the modules in the library after all other
   commands have been executed.  If you also specify the -s option, the
   listing includes the public symbols defined in each module.

@[filename]

   tells oml to execute the commands that you enter from stdin (usually
   defined as the keyboard) or from the filename, if specified.  If you
```

include this option it should be the last option on the command line.

1.30 oml options

-b

tells oml not to issue prompts, including prompts for missing information.

-n

strips debugging information from modules before adding them to the library.

-oprefix

specifies a prefix for the filenames to be created by the x command. If the prefix includes a directory name, you must include the forward slash (/) at the end of the name.

-s

includes, in the listing produced by the l command, a list of the public symbols defined in the module.

-tpathname

specifies a path for the temporary library.

-v

tells oml to display messages as it adds or deletes modules to and from the library.

-x

generates a cross reference for variables and functions used in a library.

1.31 sc

sc - SAS/C Development System driver

Synopsis

sc [options] [source-filename(s)]

Description

This is the main command used to invoke the SAS/C Compiler. It manages loading the different parts of the compiler, and generating the output based on the options you specify on the command line, and/or in an

```
   SCOptions file.

   The sc command accepts C source files, assembly source files, object
   files, and link libraries as input files.

See Also
    sc options
```

1.32 sc5

```
sc5 - SAS/C Development System V5.10 compatible driver

Synopsis

   sc5 [options] [source-filename(s)]

Description

   This version of the SC driver is provided for compatibility with older
   versions of the SAS/C Compiler.  It accepts the old-style options (such
   as -L, -cs, etc.).

   See the documentation for a version of SAS/C prior to V6 for details on
   the available options.

See Also

    sc
```

1.33 scmsg

```
scmsg - Searches for and corrects errors and warnings

Synopsis

   scmsg [ options ]

Description

   The scmsg utility helps you find and fix errors and warnings in your
   code.  It acts as a filter between you and the compiler, allowing you
   to invoke the editor of your choice and go to the line causing the
   error or warning quickly and easily.

   To use scmsg, you must compile your program with the errorrexx
   option.

   You can use scmsg with the editor of your choice, using AREXX as a
   communication medium.  AREXX is a standard part of AmigaDOS 2.0 and can
   be purchased as a third-party product for use under AmigaDOS 1.3.  If
   you do not have AREXX, you still can use scmsg to control the se editor
   provided as part of the SAS/C Development System, but you will not be
```

able to program the editor keys to control scmsg.

If you invoke the compiler from the Shell or by using the Build icon on the Workbench, scmsg is automatically invoked for you.

The current message is always highlighted. You can move from message to message by clicking on the message or by using the arrow keys. The scroll bar on the right side of the window allows you to move around in the list. Double-clicking on a message invokes the editor at the file and line number specified in the message.

Every time a C source file is compiled, all old messages listing that file as the primary file are deleted from the scmsg window.

See Also

scmsg AREXX commands , scmsg options

1.34 scmsg arexx commands

Note: The AREXX Port name for scmsg is "SC_SCMSG".

abort

aborts any builds currently running.

altfile

returns the alternate filename (if any). An empty string indicates that there are no messages on the list or that the current message has no alternate file.

altline

returns the alternate line number (if any). An empty string indicates that there are no messages on the list or that the current message has no alternate file.

bottom

goes to the last message in the list.

build [options]

calls the smake utility, using the options specified, to rebuild the last project built.

class

returns either error or warning, depending on the current message.

clear

deletes all messages.

delcomp [filename]

deletes all messages with the specified filename as their primary filename. If no filename is specified, the primary filename of the current message is used. If the current message is deleted, the next non-deleted message becomes current.

delete

deletes the current message and goes to the next message in the list.

delfile [filename]

deletes all messages with the specified filename as their secondary filename. If no filename is specified, the secondary filename of the current message is used. If the current message is deleted, the next non-deleted message becomes current.

delnum [msgno]

deletes all messages with the specified message number. If no message number is specified, the number of the current message is used. If the current message is deleted, the next non-deleted message becomes current.

file

returns the filename for the current message. An empty string indicates there are no messages on the list.

number

returns the error number. An empty string indicates that there are no messages on the list.

hide <option>

closes the scmsg window, but keeps scmsg running. By default, the scmsg window reappears when a new message is issued from the compiler. To change when the window reappears, you can specify one of the following options:

autoedit

specifies that you want scmsg to invoke your editor automatically, open the source file, and move to the line producing the message.

noautoedit

cancels autoedit mode.

rexxonly

specifies that scmsg should not open a window. Use this option if you intend to query scmsg for the messages from your editor or some other AREXX-supporting program and you do not

want to see the scmsg window. This option does not take any arguments. To cancel rexxonly mode, you must send the norexxonly command.

norexxonly

cancels rexxonly mode. If you send a norexxonly command, then scmsg opens the message browser window when it receives the next message.

You can also send a show command with AREXX or reinvoke scmsg from the command line to redisplay the window.

line

returns the line number for the current message. An empty string indicates there are no messages on the list.

next

goes to the next message in the list. Unlike Version 6.0, using the next command on the last message in the list does not move you back to the top of the list. Use the top command to go to the top of the list.

newbld <compunit>

tells scmsg to clear all messages for the specified compilation unit. When the compiler begins a new build, it sends a newbld message to scmsg. If you have set the autoedit option, newbld also forces scmsg to invoke the editor on the next new message.

newmsg "compunit" "file" line 0 "" 0 <class> <errnum> <text>

adds a message to scmsg's message list, where:

compunit
is the compilation unit associated with the message (the .c filename). Enclose the unit name in double quotes (").

file
is the filename of the message. Enclose the filename in double quotes.

line
is the line number of the message.

0 "" 0
must appear exactly as shown.

class
is either Error, Warning, Note, or Info.

errnum
is a positive error number. If errnum is zero, no error message number will be displayed.

```
   text
      is the message text.

   If the message text contains the words

       See line nnn file "filename"

   where nnn is a decimal number, then nnn and filename are
   interpreted as the alternate line number and filename associated
   with the message.

number

   is a synonym for errnum.

opts [option]

   allows you to load, set, and save scmsg options without leaving
   scmsg.  You can specify any of the following:

   load <filename>
      loads scmsg options from the specified file.

   save <filename>
      saves options to the specified file.  This file can contain
      any option that is valid on the command line or from Set
      Options in the Project menu.

   portname <name>
      sets the name of the editor's AREXX port to the specified
      name.

   screen <name>
      closes the scmsg window and reopens it on the specified public
      screen.
          the specified public screen.

prev

   goes to the previous message in the list.  If the current message is
   the first in the list, scmsg goes to the last message in the list.

quit

   terminates scmsg.

rexxonly

   specifies that scmsg should close the message browser window and not
   reopen it unless you specify scmsg nohidden at the Shell prompt or send
   the AREXX show command to the SC_SCMSG AREXX port. Use this command if
   you intend to query scmsg for the messages from your editor or some
   other AREXX-supporting program, and you do not want to see the scmsg
   window.

select
```

selects the current message for editing. Your editor is invoked and moved to the proper file and line number. This command is equivalent to pressing the Return key on the current message.

show [activate]

pops the scmsg window to the front or redisplays the window if it is hidden. If you specify activate, the scmsg window becomes the active window, and you can enter keyboard commands to scmsg.

text

returns the message text. An empty string indicates that there are no messages on the list.

top

goes to the first message in the list.

1.35 scmsg options

autoedit

specifies that you want scmsg to invoke your editor automatically, open the source file, and display the line producing the message. The default value is noautoedit.

config=filename

specifies the name of a configuration file. Typically, the configuration file contains the command necessary to invoke your editor, open a source file, and display the appropriate line. For more information about configuration files, see "Using AREXX to Invoke Your Editor," in your manual.

hidden

specifies that you do not want scmsg to display the message browser window until the compiler returns a message. The default value is nohidden.

pubscreen=<name>

tells scmsg to open its window on the public screen you specify. You can abbreviate this option as screen.

quit

terminates scmsg.

rexxonly

specifies that scmsg should not open a window. Use this option if you intend to query scmsg for the messages from your editor or some other

AREXX-supporting program, and you do not want to see the scmsg window.

1.36 scompare

scompare - Generates patch files

Synopsis

scompare [options] <oldfile> <newfile>

Description

You can use the scompare utility to create a patch file. oldfile is the name of the old binary file. newfile is the name of the new, updated binary file. Applying the patch file generated by scompare to oldfile produces a file identical to newfile. You can use the spatch utility to apply patches created by scompare.

scompare supports the following options:

-i<info-filename>
specifies the file containing messages to be displayed whenever the patch is applied.

-n<match-number>
specifies the number of bytes that must be identical to be considered a match. The default value is 10.

-o<out-filename>
specifies the name of the generated patch file. The default filename is oldfile.pch.

-a
prints the changes as text in addition to creating the patch file.

NOTE: You cannot distribute copies of scompare or any other utility, except spatch, that is included in the SAS/C Development System. The patch file that you create using scompare and the spatch utility are both freely redistributable.

1.37 scopts

scopts - Sets compiler options

Synopsis

scopts [options]

Description

The scopts utility allows you to set compiler options for a project by clicking on the gadget that corresponds to the option. The options you specify are used by the sc command whenever you compile and link your files.

To run scopts, you can double-click on the SCoptions icon, or you can enter scopts on the Shell command line. scopts displays the SAS/C Compiler Options Index screen. This window contains buttons that open additional windows. Each of these windows allows you to set different compiler options. In all, scopts can display nine windows:

> Compiler Options Index
> Compiler Options
> Diagnostic Message Options
> Code Generation Options
> Listing/Cross Reference Options
> Optimizer Options
> Prototype Generation Options
> Linker Options
> Map Options

For a brief description of an option, move the cursor to the option gadget and press the Help key.

These windows may contain one or more of the following basic types of gadgets:

> cycles

appear as raised buttons with a cycle symbol on the left side. By clicking on the gadget with the left mouse button, you can cycle through the settings available for that option. To reverse the direction of the cycle, press the Shift key as you click the mouse button.

> actions

appear as raised buttons. Selecting the button causes an immediate action to occur. For example, selecting the Compiler Options... gadget displays the Compiler Options window.

> strings

appear as a raised ridge around a text area. You can enter data in the area by clicking within the text area and typing. When you have finished typing, press the Return key.

> lists

appear as a set of control gadgets (a scroll bar, arrow gadgets, a string area, and ADD and DEL buttons) combined with a raised scrolling area. Use the scroll bar and arrow gadgets to position the scrolling area on specific items. To add new items to the list, select the ADD button, type the new item name, and press the Return key. To remove an item from the list, click on the item (to copy it to the string area), and select the DEL button. To

modify an item, click on the item (to copy it), and enter any corrections.

To specify an option for which the scopts utility does not have a gadget, enter the option and any parameters required by the option into the SPECIAL gadget. You can enter as many additional options as you like. If you use the SPECIAL gadget to specify an option for which scopts already has a gadget, then the next time you invoke scopts, the gadget for that option is selected, and the option is no longer included in the SPECIAL gadget. The only options remaining in the SPECIAL gadget are those for which scopts does not have a gadget. For example, if you enter LINK into the SPECIAL gadget, the next time you invoke scopts, the LINK gadget is selected, and the SPECIAL gadget does not contain the LINK option.

Any values you enter into the SPECIAL gadget are processed after all other gadgets are processed. Therefore, options you enter into the SPECIAL gadget may override other options.

To specify the name of your final executable module, specify a filename in the ProgramNAME gadget.

To save your option settings, select the appropriate option from the Project menu.

To exit scopts without saving any of your changes, click on the Cancel button.

When you run the sc command, it first looks for an scoptions file in your current directory. If this file does not exist, the sc command looks for the ENV:sc/scoptions file.

The scoptions file is an ASCII file that contains the list of sc options that you specify. You can edit this file with a text editor and add any options you want.

To load a previously saved options file or to reset all options to their default values, select the appropriate option from the Project menu.

You can also change option settings by specifying the option on the scopts command line. For example, to set the math=standard and link options, you can enter the scopts command as follows:

scopts math=standard link

If you enter the scopts command followed by option settings, scopts does not display any windows, and you cannot choose the file into which the options are saved. Specifically, scopts does the following:

> reads the scoptions file in your current directory, if it exists.
> If this file does not exist, scopts reads ENV:sc/scoptions.
> adds or changes the options you specified.
> saves the new option settings in the file scoptions in the current directory.

1.38 scsetup

```
scsetup - Sets up a new project

Synopsis

   sc:scsetup [options] [directory] . . .

Description

   The scsetup utility sets up a directory that you can use for C
   development.  You can run scsetup on existing directories or use it to
   create new directories. scsetup creates an icon for the directory and
   copies into the directory icons for the most commonly used SAS/C
   Development System tools, such as smake and CodeProbe.  You can add
   icons for any file extensions or tools that you want scsetup to copy in
   addition to the standard icons.  Specifically, scsetup performs the
   following actions:

      > Creates the directory, if necessary.

      > Creates an icon file for the directory being set up, if it does
  not already have one.  scsetup uses the file
  sc:starter_project.info as a template for directory icons.

      > Creates icons for the files in the directory, if necessary.
  Default icons are supplied for .c, .cxx, .cpp, .cc, .h, .a,
  and .smk files.  These icons are in the sc:icons drawer under
  the name def_c.info, def_cxx.info, def_h.info, def_a.info, and
  def_smk.info. You can add any default icons needed for other
  extensions by copying an icon into sc:icons with the appropriate
  name (def_extension.info).

      > Creates an icon file for any subdirectories of the directory being
  set up, if they do not already have one.

      > Creates icons for the files in each subdirectory, if necessary.

      > Copies the contents of the directory sc:starter_project, if
  present, to the directory being set up (if you do not
  specify the nostarter option).  This starter directory
  contains a Build icon, a Debug icon, an Edit icon, a Find
  icon, and an SCoptions icon.  You can place any other icons
  or programs in this starter directory that you want copied
  to directories set up with scsetup.

   scsetup will not overwrite any existing files or icons unless you
   specify the force option.

   You can specify the following options in the scsetup command:

      force

  tells scsetup to copy icons and files into the directory
  whether or not they already exist.  Use this option if you
  have a directory already set up for previous versions of the
```

```
  SAS/C Development System.

      nostarter

  tells scsetup not to copy the contents of the drawer
  sc:starter_project into the directory being set up.  Use
  this option to create icons for files in a directory based
  on their extension when you do not want the SCoptions, Edit,
  Debug, Build, and Find icons copied into the directory.

   You can run scsetup from the CLI or from the Workbench.  To run scsetup
   from the CLI, enter the following command:

  sc:scsetup [directory] . . .

   If you do not specify a directory name, scsetup sets up the current
   directory.  If you specify a directory that does not exist, scsetup
   creates the directory.  To run scsetup from the Workbench and create a
   new project, double-click on the SCsetup icon, and type the name of the
   drawer when prompted.  To run scsetup from the Workbench and set up an
   existing drawer, click on the drawer icon. (Click on the actual drawer
   icon, not the file icons in the drawer.)  To select additional drawers,
   hold down the Shift key and click on their icons.  After you have
   selected all the drawers, hold down the Shift key and double-click on
   the SCsetup icon.

   After you have set up a directory, you can run the editor by
   double-clicking on the Edit icon, and you can run CodeProbe by
   double-clicking on the Debug icon.  You can run smake by
   double-clicking on the Build icon.  smake looks for a smakefile or a
   makefile.  If none are present, it compiles and links all .c files in
   the directory using the options specified through the scopts utility,
   described earlier in this chapter.
```

1.39 slink

```
slink - SAS/C Development System Linker

Synopsis

   slink object-files [ options ]

Description

   This program takes the object files created by using the SC command,
   and creates the final executable file.

   It will be invoked automatically from SC if you specify the  LINK
   option to the SC command.

See Also

    slink options
```

1.40 slink options

addsym
bufsize
define
fast
from
height
indent
libprefix
librevision
map
newocv
nodebug
onedata
ovlymgr
plain
quiet
swidth
smalldata
to
verify
with
batch
chip
fancy
faster
fwidth
hwidth
libfd
library
libversion
maxhunk
noalvs
noicons
overlay
prelink
pwidth
root
smallcode
stripdebug
verbose
width
xref

1.41 addsym

addsym
generates hunk_symbol records for all externally-visible symbols in each object file and library module whether or not the object file was compiled with the debug compiler option. This option gives CodeProbe the location, but not the size or type, of any externally visible symbols in your program.

1.42 batch

batch
sets the value of all undefined data symbols to 0 and all undefined code symbols to __ _stub. Normally, slink asks you to enter a value for each undefined symbol. However, if you specify batch, slink does not prompt you to enter values for undefined symbols. If an undefined function is called, the library function __stub (__ _stub to the linker) is called instead. The library's version of __stub displays a requester telling you that an undefined routine was called and allows you to choose whether to abort or continue.

1.43 bufsize

```
bufsize <number>
   sets the I/O buffer size for slink. By default, slink buffers all
   object modules.  Buffering requires more memory but reduces the
   time required to link your program.  If you run out of memory
   while linking, try linking with bufsize 4096.  This specification
   tells slink to buffer only one file at a time and to use a buffer
   size of 4096 bytes.
```

1.44 chip

```
chip
   specifies that all hunks are to be placed in chip memory
   regardless of the input object hunk specifications.  However, if
   you specify fast or chip for the datamem, codemem, or bssmem
   compiler options, that value overrides this option.

   The chip option is provided for compatibility with previous
   versions of the linker.  It is recommended that you use the
   datamem, codemem, and bssmem compiler options instead.
```

1.45 define

```
define <symbol[=value]>
   defines a symbol to be used in the linking process. You can use
   this option with the prelink option to force specific routines
   from the libraries to be linked into your program even though
   your program does not contain any references to the routines.

   Remember that linker symbols have an extra underscore added to
   the beginning of the symbol name.  To refer to a function foo
   using the define option, you must specify _foo.  To refer to
   registerized functions, add an at (@) sign to the beginning of
   the symbol name, as in @foo.

   NOTE:  Do not confuse this option with the define compiler
    option.
```

1.46 fancy

```
fancy
   enters control characters to highlight and bold headings in the
   map file.  The option is on by default. To disable the use of
   these control characters, specify the plain option.  fancy is
   ignored if you do not specify the map option.
```

1.47 fast

```
fast
   specifies that all hunks are to be placed in fast or expansion
   memory regardless of the input object hunk specifications.
   However, if you specify fast or chip for the datamem, codemem, or
   bssmem compiler options, that value overrides this option.

   The fast option is provided for compatibility with previous
   versions of the linker.  It is recommended that you use the
   datamem, codemem, and bssmem compiler options instead.
```

1.48 faster

```
faster
   is included only for compatibility with the Commodore linker,
   alink.
```

1.49 from

```
from <object-filename(s)>
   identifies the object files that are the primary input to the
   linker.  You can specify the from option as many times as
   necessary in the slink command.  If you specify the object files
   as the first items in the slink command, you do not need to use
   the from option.  The root option is a synonym for this option.

   The object files are copied to the root of the object module.
   You must specify at least one object file in the slink command.
```

1.50 fwidth

```
fwidth <number>
   specifies the filename width, in columns, in the map file.  The
   default value is 16.  This option is ignored if you do not
   specify the map option.
```

1.51 height

```
height <number>
   specifies the total number of lines on a page in a map file.  If
   you specify 0, the linker does not add form feed characters to
   the file.  The default value is 55. This option is ignored if you
   do not specify the map option.
```

1.52 hwidth

```
hwidth <number>
   specifies the hunk name field width, in columns, in the map file.
   The default value is 8.  This option is ignored if you do not
   specify the map option.
```

1.53 indent

```
indent <number>
   specifies the number of columns to indent on a line in the map
   file.  The map file is indented n columns from the value
   specified by the width option.  The default value is 0.  This
   option is ignored if you do not specify the map option.
```

1.54 libfd

```
libfd <filename>
   specifies the name of a function description (.fd) file.  Use
   this option only if you are building a shared library.

   For information on creating shared libraries, refer to SAS/C
   Development System Library Reference.
```

1.55 libprefix

```
libprefix <prefix>
   specifies the prefix you want added to the functions listed in
   the function description (.fd) file.  The default prefix is an
   underscore (_).  Use this option only if you are building a
   shared library.  The libprefix option allows the library to call
   entry points within itself, without using the library base.

   Make sure that the function declarations in your source code
   match the full name, including any specified prefix.  For
   example, suppose your library has a function called myfunc.  Your
   .fd file contains the following line:

      myfunc(a)(d0)

   If you specify a prefix of _LIB, you should declare the function
   in your source code as LIBmyfunc, as shown in the following
   example:

      #pragma mylibbase myfunc 1e 001

      __asm LIBmyfunc(register __d int a)
      {
   return a+1;
```

```
      }

      void test(void)
      {
   LIBmyfunc(1);   /* Call myfunc directly.                 */
   myfunc(2);      /* Call myfunc through the library base. */
      }
```

For information on creating shared libraries, refer to SAS/C Development System Library Reference.

1.56 library

library <link-library-filename(s)>
specifies the library files that you want the linker to search for modules referenced in your code. Only referenced modules from library files are included in the final executable module. You can abbreviate this option as lib.

1.57 librevision

librevision <number>
specifies a minor revision number of the library you are creating. If you do not specify a revision number, it defaults to 0. Use this option only if you are building a shared library.

For information on creating shared libraries, refer to SAS/C Development System Library Reference.

1.58 libversion

libversion <number>
sets the version number of the library you are creating. You can set the version number to any number from 0 to 255. The default is 1. Use this option only if you are building a shared library.

For information on creating shared libraries, refer to SAS/C Development System Library Reference.

1.59 map

map <[mapfile[map-option[,] map-option...]]>
tells slink to create a map file. The map file contains information on the size and location of all hunks and the value of all symbols.

```
   If you do not specify a mapfile, the linker writes the map
   information to the file executable.map.  If you specify a
   mapfile, you can also specify the following map file options:

      f   lists the hunks in each file
      h   lists hunks by size and originating function
      l   lists hunks by library function
      o   lists hunks in each overlay
      s   lists where symbols are defined
      x   creates a symbol cross-reference that lists where the
    symbols are defined and their definition.
```

1.60 maxhunk

```
maxhunk <n>
   limits to n bytes the size of the hunks that slink creates when
   merging hunks. By default, there is no limit on hunk size.
```

1.61 newocv

```
newocv
   This option tells slink to use the new overlay manager.  The new
   overlay manager will allow you to call registerized parameter
   functions across overlays.  The name of the new overlay manager
   is @ovlyMgr. The old overlay manager is called _ovlyMgr.
```

1.62 noalvs

```
noalvs
   issues warning messages if slink generates an automatic link
   vector (ALV) instruction to resolve 16-bit program
   counter-relative references.  Using this option stops slink from
   creating a non-relocatable module from what was intended to be
   relocatable code.
```

1.63 nodebug

```
nodebug
   is included for compatibility with previous releases of the
   linker.  Use the stripdebug option instead.  You can abbreviate
   this option as nd.
```

1.64 noicons

```
noicons
   prevents the linker from creating icons for the files that it
   creates.
```

1.65 onedata

```
onedata
   tells the linker to merge all data, bss, and chip sections.
   Merging hunks may decrease the time required to load your program
   but may produce larger hunks that are difficult to scatter load.
```

1.66 overlay

```
overlay
   identifies the start of an overlay tree.  You should enter the
   overlay option and the list of files in each overlay in a with
   file, as follows:

   overlay
   overlay-1-filename(s)
   *overlay-2-filename(s)
   .
   .
   .
   #
   You must terminate the overlay tree with a line consisting of a
   single pound (#) sign.  For more information, see " Using
   Overlays," later in this chapter.  See also the description of
   the with option.
```

1.67 ovlymgr

```
ovlymgr <filename>
   tells slink to use the filename you specify as the overlay
   manager instead of the default filename, ovs.o, which is
   contained in the libraries.  The file you specify should consist
   of a single code hunk named NTRYHUNK and define the global symbol
   _ovlyMgr.
```

1.68 prelink

```
prelink
   tells slink to produce an object module with symbol references
   and definitions still intact. You can then link with this object
   module to produce an executable module.  This option is useful if
   you are developing a large application and changing only a single
   source module.  Do not specify this option if you are using
   overlays.
```

1.69 plain

```
plain
   turns off the fancy option.  This option tells the linker not to
   enter control characters for highlighting and bold in the map
   file.  This option is ignored if you do not specify the map
   option.
```

1.70 pwidth

```
pwidth <number>
   specifies the width of program unit name field in the map file.
   The default value is 8.  This option is ignored if you do not
   specify the map option.
```

1.71 quiet

```
quiet
   suppresses all linker messages except error messages.
```

1.72 root

```
root <object-filename(s)>
   specifies the object files that are the primary input to the
   linker.  These object files are always copied to the root of the
   object module.  You must specify at least one object file for the
   root.  If the primary input files appear as the first option to
   slink, then the root keyword is optional and may be omitted. The
   from option is a synonym for root.  You can specify the root
   option more than once.  The files you specify with root are added
   to the list of files to be linked.
```

1.73 swidth

```
swidth <number>
   specifies the width of the symbol names field in the map file.
   Default value is 8.  This option is ignored if you do not specify
   the map option.
```

1.74 smallcode

```
smallcode
   tells the linker to merge all code hunks.  You can abbreviate
   this option as sc.
```

1.75 smalldata

```
smalldata
   tells the linker to merge all data and bss sections. Merging
   hunks may decrease the time required to load your program, but
   may produce larger hunks that are difficult to scatter load. You
   can abbreviate this option as sd.  The smalldata option does not
   merge chip data with non-chip data.  To merge chip data also,
   specify the onedata option instead.
```

1.76 stripdebug

```
stripdebug
   suppresses any H_DEBUG and H_SYMBOL debug information in the
   final executable file.
```

1.77 to

```
to <filename>
   specifies the name of the final executable module. If you do not
   specify the to option and you specify one filename on the slink
   command line, slink will use that filename (without the file
   extension) as the object module name.  If you specify more than
   one filename, slink assumes that the first filename is the name
   of a startup module and will use the second filename as the
   object filename.
```

1.78 verbose

```
verbose
   prints the names of each file as the file is processed.
```

1.79 verify

```
verify <filename>
   specifies a file to which you want the linker to write all
   messages.  If you do not specify the verify option, the linker
   writes all messages to stdout.  You can abbreviate this option as
   ver.
```

1.80 width

```
width <number>
   sets the maximum line length for the map and cross reference
   listing files. The default value is 80. This option is ignored if
   you do not specify the map option.
```

1.81 with

```
with <filename>
   specifies a file containing slink options. The linker processes
   the contents of with files as if you had specified the option in
   the slink command.  You can specify the with option as many times
   as necessary.  Also, you can enter with options in a with file.

   The following is an example with file.

      from lib:c.o vt100.o init.o window.o xmodem.o remote.o
   kermit.o script.o
      library lib:sc.lib lib:amiga.lib
      verbose
      smallcode
      smalldata
      to vt100

   NOTE:  Do not confuse this option with the with compiler option.
```

1.82 xref

```
xref <filename>
   specifies a file to which you want the linker to write cross
   reference information. If you do not specify xref, but you
   specify the map option, the linker writes the cross reference
   information in the map file.

   NOTE:  Do not confuse this option with the xref compiler option.
```

1.83 smake

```
smake - Maintains and updates records of file dependencies

Synopsis

   smake [ options ] [macro-definitions] [targets]

Description

   The smake utility is a tool that you can use to maintain projects that
   are composed of many files.  A file can be C source code, a data file
   for a graphics or audio, or perhaps a spreadsheet file.  For example,
   you may have a project that consists of 50 files, and several
   programmers may be responsible for different files.  To manage this
   project, you need to keep track of which files have been changed and
   which files must be compiled before other files (that is, which files
   are dependent on other files).  You can use smake to keep track of file
   dependencies, recompile and relink any files that have been updated,
   and produce new product files.

   In other words, smake determines if any of the source files have
```

changed since the last version of the product file was generated and, if so, generates a new product file.

To use smake, you create a smakefile that identifies dependent files and target files and describes the actions required to produce a new product. A basic description of these terms follows:

> target file

is any file that is created or updated by smake. Producing this file may require creating or updating several intermediate files.

> dependent file

is a file that must be created or updated before a target file can be updated. A dependent file can be a source code file, header file, or object code file. In other words, if a dependent file is changed (for example, by editing), then the target file must be rebuilt.

> actions

are the commands necessary to update each dependent and target file. Actions can be calls to the compiler or linker or basic housekeeping commands.

> smakefile

is a file that identifies every dependent file required to create the target file and describes every action required to update or create the dependent and target files.

NOTE: You can use the mkmk utility to generate smakefiles.

When you run smake, it re-creates only the first target file specified in the smakefile (unless you specify an alternate target as described in "Using Alternate Targets," later in this section) and, if necessary, any dependent files needed to re-create the target file. Specifically, smake performs the following actions:

> locates and identifies the target file.

By default, smake remakes the first target file specified in the smakefile. You can specify an alternate target as described under "Using Alternate Targets," later in this section.

> ensures that any file on which the target file depends already exists and is up to date.

To determine when dependent files were last modified, smake looks at the time stamp of the file. A file is considered current if it has a time stamp later than that of any of its dependent files.

NOTE: Before using smake, make sure that your system clock is accurate, using the date command if appropriate.

An example of the date command under AmigaDOS follows:

```
date 12-Aug-92 16:34:57
```

You must enter the time using 24-hour clock notation. Enter the month as a three-character field.

> Re-creates the target file if any of the dependent files have been modified more recently than the target file.

For details on creating a smakefile, please see your manual.

Running smake

You can run smake from the Workbench or from the CLI.

To run smake from the Workbench, double-click on the Build icon. smake looks for the smakefile in your current directory. The default file names that smake looks for, in the order in which it looks for them, are as follows:

```
smakefile

smakefile.smk

lmkfile

lmkfile.lmk

makefile
```

To use a different smakefile name, specify the -f option as described later in the "smake options" section. To specify options, define macros, or specify alternate targets for the Build icon, select the Build icon, choose Information from the Icons menu, and add the necessary parameters in the Tool Types box.

When running smake from the Workbench, the search order of files is based on the smake_files: assignment.

You can create the assignment using the following command (from the CLI):

```
assign smake_files: your-smake-file-location
```

When no assignment has been made, smake searches the current directory for the required files.

To run smake from the CLI, enter the smake command as follows:

```
smake [options] [macro definitions] [targets]
```

As with the Build icon, smake looks for one of the three default smakefile names in your current directory. To use a different smakefile name, specify the -f option as described in the smake options.

```
See Also
    smake options
```

1.84 smake options

```
-a

   rebuilds all targets and subtargets without regard to time stamps.

-bfile

   uses the .def file you specify instead of the default file smake.def.

-c

   tells smake to record everything it would have done if it had executed
   the command you entered.  When you specify the -c option, smake does
   not execute the command you enter.  Instead, it creates a batch file
   containing all the instructions it would have executed if you had not
   specified the -c option.  This file is named smakefile.bat in the
   current directory, and you can run it by entering the following on the
   command line:

  execute smakefile.bat

-d

   prints detailed debugging information about the processing of the
   smakefile.

-e

   erases any out-of-date targets before remaking them.

-f<file>

   uses the filename you specify as the input smakefile.  smake attempts
   to locate the file with the exact name you have specified.  If this
   search is unsuccessful, smake searches for a file with the name you
   specified plus an extension of .smake.

-h

   prints help information and then exits.

-i

   ignores errors caused by executing the actions.  Both the -k and -i
   actions should be used with caution because smake will continue
   running. The -k option incorporates all of the functionality of the -i
   option.

-k

   ignores error returns from actions passed to AmigaDOS and from smake
```

not knowing how to make certain targets. Both the -k and -i actions should be used with caution because smake will continue running. The -k option incorporates all of the functionality of the -i option.

-n

displays the actions smake would have taken, but smake does not execute these actions.

-p

prints target descriptions and expanded macros.

-q

determines if the target file is current. smake prints a 1 if the target is current or a 0 if it is not. smake will not execute any actions.

-s

does not echo actions to the screen before executing them.

-t

touches the target files by updating them with the current system time. smake does not execute any of the actions associated with these targets.

-u

rebuilds unconditionally all targets and subtargets without regard to time stamps.

-w

forces smake to act as if it was invoked from Workbench.

-x

is for UNIX compatibility. If you specify this option, smake attempts to detect any features of the smakefile that would prevent it from working correctly with the UNIX make utility.

1.85 smfind

smfind - Finds strings in text files

smfind <pattern> <file> [file]. . .

Description

The smfind utility helps you find character strings in your source code. smfind uses the grep utility to locate the character string you specify in the files that you specify, and

```
   it displays each occurrence of the character string using the
   message browser (scmsg).  You can then double-click on one or
   more occurrences of the character string, and scmsg will open the
   appropriate file at the correct location.

   You can run smfind from the Shell or the Workbench.

   If smfind finds any matches in the files that you specify, it
   invokes scmsg and sends all occurrences to scmsg.  You can then
   double-click on each occurrence to move to that file and line
   number.
```

1.86 spatch

```
spatch - Applies patches to files

Synopsis

spatch [options] <oldfile>

Description

   The spatch utility applies patches to files.  A patch file is a
   binary file that describes the differences between an existing
   file and a new version of that file.  You can use the scompare
   utility to generate patch files.

   <oldfile> is the name of the file to be patched.

   spatch supports the following options:

     -o<out-filename>
  is the name of the generated patched file.  If you do not
  specify the -p option, spatch looks for the file oldfile.new
  in the current directory.

     -p<patchfile>
  is the name of the patch file generated by the scompare
  utility.  The default file is oldfile.pch in the current
  directory.

   NOTE:  You cannot distribute copies of scompare or any other
    utility, except spatch, that is included in the SAS/C
    Development System.  The patch file that you create using
    scompare and the spatch utility are both freely
    redistributable.
```

1.87 splat

```
splat - Searches for and replaces patterns that match regular expressions

Synopsis
```

```
   splat [ options ] <pattern> <string> <file> . . .

Description

   splat replaces all occurrences of pattern with string in each file that
   you specify.  splat does not overwrite the original version of the file
   unless you specify the -o option.  If you do not redirect the output by
   specifying a greater than (>) sign followed by a destination, splat
   places the changed file in either a temporary file or, if you use the
   -d option, in the specified directory.  splat uses the same root
   filename with an extension of .$$$.

   You must use the rules recognized by grep for specifying the pattern
   and string.  For example, to specify a string that contains spaces, you
   must enclose the string in double quotes.  For additional information,
   see the description of the grep utility earlier in this chapter.

See Also

    splat options
```

1.88 splat options

```
-ddirectory

   specifies the directory where you want splat to place the new files.
   Using this option leaves the original versions untouched.  If the
   specified directory does not exist, splat attempts to create it.  If
   the attempt fails, splat displays an error message.  The -o and -d
   options are mutually exclusive.

-o

   tells splat to overwrite the original version of the file with the new
   version.  The -o and -d options are mutually exclusive.

-s

   displays the name of the file on which splat is currently working.  It
   also displays a message saying that no substitutions have been
   performed if it was unable to find the specified pattern in the file.
   If you do not specify -s, splat performs its task silently.

-v

   displays all lines in which substitutions are made, but it does not
   create new versions of the files.  You can use this option to determine
   the changes that splat will make to your files.
```

1.89 sprof

sprof - Determines the time a program spends in each function

Synopsis

```
sprof [report n] [program [program-options]]
```

Description

The sprof utility tells you how much time your program spends executing each function and how many times each was called. In other words, it profiles your code. Profiling tells you where your code spends most of its time. You can use this information to identify areas of your program where efficiency can be improved. sprof has these advantages over the other SAS/C profiler, lprof:

- sprof tells you exactly how much time was spent in each function.
- sprof tells you how many times each function was called.
- sprof filters out the time spent waiting for the user, if you use the appropriate calls to PROFILE_ON and PROFILE_OFF. These macros are described below.
- You can use sprof from a library, device, file system, task, the Workbench, or any other way of invoking an application on the Amiga.

This utility works under AmigaDOS version 2.0 or later. It does not work under AmigaDOS version 1.3.

The report option tells sprof to produce intermediate reports on the status of the program. The parameter n is an integer that specifies the number of seconds between reports. You can also get a report at any time by sending a Control-F signal to the sprof process. If you invoked sprof from the Shell, type Control-F in the Shell in which sprof is running. If you invoked sprof from the Workbench, there is no easy way to send a Control-F to sprof.

To use the profiler, you must first compile your program with the profile option.

You can run sprof from either the command line or the Workbench.

1.90 guiprof

guiprof - Illustrates time a program spends in each function

Synopsis

```
guiprof <prof options> [program [program options]]
```

Description

Similar to sprof, the guiprof utility tells you how much time your program spends executing each function and how many times each was called. Unlike sprof, however, guiprof displays this information as a dynamic histogram. You can use this information to identify areas of your program where efficiency can be improved. guiprof works the same way as sprof, by using special compiler-generated hooks in the entry and exit code for each function. For more information on using guiprof, see the READ.ME file in the directory SC:EXTRAS/GUIPROF.

To use the profiler, you must first compile your program with the profile option.

You can run guiprof from either the command line or the Workbench.

1.91 tb

tb - Displays traceback information

Synopsis

```
tb [>destination] [ options ] [tbfile [program]]
```

Description

The tb utility processes the traceback file that is created when you link your program with catch.o and your program terminates abnormally. To get traceback information for your program, you must link your program with catch.o. If your program terminates abnormally, catch.o creates a standard IFF format file named Snapshot.TB in your current directory. This file contains up to six sections:

symbol cross reference
stack
registers
environment
memory
user data.

Each section contains information about your program at the time it terminated. (Two sections, memory and user data, may not be available for your program.) tb displays the traceback information on the screen unless you redirect it by specifying a greater than (>) sign followed by a destination. If you do not specify any options, tb displays only the current stack frame, which indicates the location where the program terminated. By specifying options, you can print all of the sections in the traceback file.

By default, tb looks for the traceback file Snapshot.TB in your current directory. If you want tb to use a different traceback file, specify the traceback file as the tbfile parameter.

The program parameter specifies the program from which you want tb to

```
   read debugging information.  If you do not specify program, this
   utility uses the program specified in the traceback file.

   NOTE: Specifying a program is useful when you have two executables of
   your program: one created with debugging information and another
   without debugging information.  The version without debugging
   information loads faster, but you still have access to the
   traceback information if your program crashes.

See Also
    tb options
```

1.92 tb options

```
-l

   displays all sections present in the traceback file

-x

   displays the location of all symbols encountered in the program

-s

   displays the contents of the entire stack at the time your program
   terminated

-r

   displays the current contents of registers at the time your program
   terminated

-v

   displays the callback chain and the location where the program
   terminated

-m

   displays the amount of memory available at the time your program
   terminated (if present in Snapshot.TB)

-u

   displays any user data generated by your program (if present in
   Snapshot.TB).
```

1.93 HELP

```
You have reached this Help window by either clicking on the Help
button or by hitting the Help key within the SAS/C Help utility.
Unlike other help topics present in the SAS/C Help utility, the Help
```

help topic opens its own window. You must close this window by clicking on the close gadget or hitting escape before returning to the SAS/C help utility. You cannot hit the Retrace button to return.

To quit the SAS/C Help utility, select Quit from the Project menu or click on the close gadget. You may also hit escape.

Most help screens will display one or more buttons as part of the text. Clicking on these buttons will provide further information on the topic listed on the button. You can also reach these help topics through the main Contents screen or one of its sub-screens.

In addition, double-clicking in the help window will bring up a help screen for the word under the mouse cursor, if such a help screen exists.

While in the SAS/C Help utility, you may retrace your steps through the help screens you have selected by clicking on the Retrace button.

The Browse buttons will move you forward and backwards between help screens. The help screens are usually arranged alphabetically by command or topic.