sc

COLLABORATORS

	TITLE : sc	

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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

sc

1.1 sc.guide

MapSymbols
Math
MaximumWarnings
Modified
MultipleIncludes
ObjectLibrary
OldPreprocessor
Optimize
OptimizerComplexity
OptimizerGlobal
OptimizerInLocal
OptimizerPeephole
OptimizerSize
OptimizerSchedule
Precision
PreprocessorOnly
ProgramName
ResetOptions
ShortIntegers
SmallData
StackCheck
StandardIO
Strict
StringMerge
StringSection
To
Underscore
UtilityLibrary
Version
WarnVoidReturn
XREF
XReferenceHeaders
MapXreference
MaximumErrors
MemorySize
MultipleCharacterConstants
Object
ObjectName
OnError
OptimizerAlias
OptimizerDepth
OptimizerInline
OptimizerLoop
OptimizerRecurDepth
OptimizerTime
Parameters
PreprocessorBuffer
Profile
PubScreen
Saveds
SmallCode
SourceIs
StackExtend
StartUp
StringsConst
StripDebug
StructureEquivalence
Trigraph
UnsignedChar
Verbose
Warn
With
XReference
XReferenceSystem

1.2 mainpanel

GenProtos
Link
List
Map
Optimize
ProgramName
SPECIAL
Verbose
Version
Xref

Save
Save Default
Cancel

1.3 compilerpanel

```
CommentNest                     aCxxOnly
Debug                           Define
GST                             GSTImmediate
Icons                           IncludeDirectory
MakeGST                         MemSize
Modified                        MultipleCharacterConstants
MultipleIncludes                PreProcessOnly
ShortIntegers                   StringsConst
StringMerge                     UnsignedChar
WarnVoidReturn

OK
```

1.4 messagepanel

```
ANSI                            Error
ErrorConsole                    ErrorHighlight
ErrorList                       ErrorRexx
ErrorSource                     Ignore
MaximumErrors                   MaximumWarnings
OnError                         PubScreen
Strict                          StructureEquivalence
Warn

OK
```

1.5 codepanel

```
AbsFuncPointer                  AutoRegister
BSSName                         Code
CodeName                        Common
ConstLibBase                    Coverage
CPU                             Data
DataMemory                      DataName
DisAssemble                     LibCode
Math                            ObjectLibrary
ObjectName                      Parameters
Precision                       Profile
Saveds                          SourceIs
StackCheck                      StackExtend
StringSection                   UtilityLibrary

OK
```

1.6 listpanel

```
ErrorList                       List
ListFile                        ListHeaders
ListIncludes                    ListMacros
```

```
ListNarrow                          ListSystem
XReference                          XReferenceHeaders
XReferenceSystem

OK
```

1.7 optimizerpanel

```
Optimize                            OptimizerAlias
OptimizerComplexity                 OptimizerDepth
OptimizerGlobal                     OptimizerInline
OptimizerInLocal                    OptimizerLoop
OptimizerPeephole                   OptimizerRecurDepth
OptimizerSize                       OptimizerTime
OptimizerSchedule

OK
```

1.8 prototypepanel

```
GenProtoDataItems                   GenProtoExterns
GenProtoFile                        GenProtoParameters
GenProtos                           GenProtoStatics
GenProtoTypedefs

OK
```

1.9 linkerpanel

```
AddSymbols
Batch
CheckAbort
Custom
LibCode
LibFD
LibPrefix
Library
LibRevision
LibVersion
Link
LinkerOptions
SmallCode
SmallData
StartUp
StripDebug

OK
```

1.10 mappanel

```
Map
MapFile
MapHunk
MapLib
MapOverlay
MapSymbols
MapXreference

OK
```

1.11 save

```
Clicking over this gadget will create (or update) a file named SCOPTIONS
in the current directory.
```

1.12 save default

```
Clicking over this gadget will create (or update) a global SCOPTIONS file,
which resides in the ENV:SC directory.
```

1.13 cancel

```
Clicking over this gadget will cancel any changes made, and will exit the
SCOPTS program without modifying any files.
```

1.14 special

```
This string gadget may be used for any option not directly supported
through gadgets in the SCOPTS program.
```

1.15 ok

```
Accept changes on this subpanel and return to Main panel.
```

1.16 custom

```
This string gadget is used to enter the name of a user-created startup
file.  This gadget is only active if  STARTUP =<user>.
```

1.17 absfuncpointer

AbsFuncPointer

generates 32-bit references to functions when loading function pointers. The default value is noabsfuncpointer. The minimum acceptable abbreviation is afp.

If you specify noabsfuncpointer and you take the address of a function that is more than 32k away, the linker generates an ALV (automatic link vector) jump instruction that allows your code to work. However, if you compare the address of the function to a function pointer assigned elsewhere, you may get a different value. Specify absfuncpointer if your code hunk is larger than 32K or in multiple code hunks and you compare function pointers. Specifying absfuncpointer may increase the size of your executable.

1.18 addsymbols

AddSymbols

tells the linker to add symbol information to the executable module. The default value is noaddsymbols. The minimum acceptable abbreviation is addsym.

This option is automatically enabled if you specify the debug option. This option is ignored if you do not specify the link option.

For more information, refer to the description of AddSymbols in the linker options documentation.

1.19 ansi

ANSI

enforces the strictest interpretation of the ANSI C standard. The default value is noansi.

Specifying ansi does the following:

> suppresses warning 95 , promotes warning 193 to an error, and produces many additional warning messages. For a complete list of messages that are enabled by the ansi option, see "Enabling Suppressed Messages," in Appendix 2.

> prevents the definition of certain preprocessor symbols. For a list of these symbols, see "Using Preprocessor Symbols Defined by the Compiler," in the Compiler Options chapter.

> disallows the initialization of variables declared with the extern keyword. For example, the following statement is not

```
   allowed:

        extern int i = 7;

>  disables the far, near, and chip keywords.  However, __far,
   __near, and __chip are still valid.

To be completely  ANSI -compliant (that is, if you require a pure  ANSI
namespace), you should also define the preprocessor symbol
 _STRICT_ANSI  to 1 before including any header files.  For more
information about  _STRICT_ANSI , refer to Chapter 7, "C Library
Reference," in SAS/C Development System Library Reference.  If you
require support for  ANSI  trigraphs, specify the  trigraph  option
also.

For more information about improving the portability of your code,
see Chapter 13, "Writing Portable C Code."

This option is ignored when compiling C++ files.
```

1.20 argumentsize

```
ArgumentSize=n

sets the size of the maximum argument to a C preprocessor macro.
The minimum acceptable abbreviation is argsiz.  This option does not
have a negative form.  The default value is 512.  However, the
memorysize option sets the argumentsize (and other internal limits)
to different default values if you do not specify argumentsize.

See the description of the memorysize option for more information.

This option is ignored when compiling C++ files.
```

1.21 assembler

```
Assembler=filename(s)

specifies assembly-language files that are to be assembled and, if
you specify the  link  option, linked into the program.  The minimum
acceptable abbreviation is asm.  This option does not have a
negative form.

You can use AmigaDOS wildcard characters to specify filenames.  To
specify several filenames or wildcard patterns, separate each
filename with a plus (+) sign or a comma (,). You can specify the
assembler option as many times as necessary.

If you are assembling a disassembly that was generated with the
disassemble compiler option, specify the underscore compiler option
also.  (If you assemble a disassembly by calling the assembler
directly, specify the -u assembler option.)
```

See also the descriptions of the csource , object , and library options.

1.22 autoregister

AutoRegister

enables automatic register selection by the code generator. The default value is autoregister. The minimum acceptable abbreviation is autoreg.

If you specify autoregister, the compiler attempts to add register variables to the variables that have already been chosen by the global optimizer or declared with the register keyword.

1.23 batch

Batch

tells the linker not to prompt for definitions of undefined symbols. The default value is nobatch. This option is ignored if you do not specify the link option. See also the description of the batch linker option.

1.24 bssmemory

BSSMemory=type

specifies the type of memory into which uninitialized external data items should be loaded. You can specify any, chip, or fast for type. You can abbreviate these values as a, c, or f. The default value is any. This option does not have a negative form.

This option affects code generated by both the compiler and the assembler. See also the descriptions of the datamem and codemem options.

1.25 bssname

BSSName=name

names the uninitialized data section. The default value is udata. You can specify bssname=none or nobssname if you want an unnamed BSS section. The linker automatically merges all sections with the same name. See also the descriptions of the codename and dataname

options. See Chapter 12, "How the Compiler Works" for information on data and code sections.

1.26 buildproject

BuildProject

rebuilds the current project. The minimum acceptable abbreviation is bldprj. This option does not have a negative form.

This option compiles and links the source files specified on the command line or in the scoptions file. This option turns on the modified , link , and errorrexx options.

Entering sc buildproj from the Shell is equivalent to entering:

```
sc #?.c #?.a #?.cxx #?.cpp #?.cc modified errorrexx link
```

Entering sc filename buildproj is equivalent to entering:

```
sc filename modified errorrexx link
```

1.27 checkabort

CheckAbort

enables Control-C in your program. The default value is checkabort. The minimum acceptable abbreviation is chkabort. This option is ignored if you do not specify the link option.

If you compile with checkabort, your program will check for Control-C interrupts whenever I/O is performed. If you specify nocheckabort, the compiler passes the following option to the linker:

```
define @__chkabort=@__dummy
```

dummy is a function provided in the link libraries that simply returns. Specifying nocheckabort makes all calls to the __chkabort function do nothing, thereby removing Control-C checking.

Do not specify nocheckabort if you are redefining the library function __chkabort in your own code.

1.28 code

Code=reference-type

specifies whether you want 16-bit or 32-bit references to functions not declared in the current file. You can specify near or n for

16-bit references or far or f for 32-bit references. The default value is near. This option does not have a negative form.

Most programs do not need this option even if they are very large, because the linker creates a jump instruction (an ALV) for any references to functions that are out of range. See also the description of the data option.

1.29 codememory

CodeMemory=type

specifies the type of memory into which code should be loaded. You can specify any, chip, or fast. You can abbreviate these values as a, c, or f. The default value is any. This option does not have a negative form.

This option affects code generated by both the compiler and the assembler. See also the descriptions of the bssmem and datamem options.

1.30 codename

CodeName=name

names the code section. The default value is text. You can specify codename=none or nocodename if you want an unnamed code section. The linker automatically merges all sections with the same name. See also the descriptions of the bssname and dataname options. See Chapter 12, "How the Compiler Works" for information on data and code sections.

1.31 commentnest

CommentNest

allows nested comments. The default value is nocommentnest. The minimum acceptable abbreviation is cnest.

Nested comments occur when one comment is totally contained inside another. The ANSI Standard prohibits nested comments, so in ANSI -compliant code, the first comment end sequence (*/) terminates both comments. For example, the statement below generates an error with nocommentnest, but compiles successfully with commentnest.

```
   /* i = i+1; /* This is a comment */ */
```

The statement below compiles successfully with nocommentnest, but generates an error with commentnest.

```
/* i = i+1; /* This is a comment */
```

1.32 common

Common

tells the compiler to use the relaxed reference-definition model for external data. If you specify nocommon, the compiler uses the strict reference-definition model. The default value is nocommon.

The section "Using Relaxed Ref-Def Model External Data" in Chapter 11, "Using SAS/C Extensions to the C and C++ Languages," describes relaxed and strict reference-definition models.

1.33 constlibbase

ConstLibBase

tells the compiler that library base pointers are set once then remain constant throughout your entire program. The default value is constlibbase. The minimum acceptable abbreviation is constlib.

If you change the value of your library bases after setting them, you should specify the noconstlibbase option. Specifying constlibbase allows the compiler to prevent extra register loading when making a series of calls to library functions through an external variable containing the library base.

1.34 coverage

Coverage

tells the compiler to generate code to collect coverage analysis information. The default value is nocover. The minimum acceptable abbreviation is cover.

Coverage analysis information allows you to determine which lines of your program have been executed by your test cases. For more information, refer to the description of the cover utility in SAS/C Development System User's Guide, Volume 2: Debugger, Utilities, Assembler.

Unlike Version 6.0, the necessary coverage initialization and termination routines are in the standard link library. You do not have to link with a special object file.

1.35 cpu

CPU=processor

generates code specific to the specified processor. You can specify any, a, or 68000 to generate code for any processor. You can also specify 68010, 68020, 68030, or 68040 to generate code for a specific processor. The default value is any. This option does not have a negative form.

This option defines one or more preprocessor symbols. See the section "Using Preprocessor Symbols Defined by the Compiler," later in this chapter for a list of those symbols.

This option affects code generated by both the compiler and the assembler.

1.36 csource

CSource=filename

specifies C source files that are to be compiled and, if you specify the link option, linked into the program. The minimum acceptable abbreviation is csrc. This option does not have a negative form.

You can use AmigaDOS wildcard characters to specify filenames. To specify several filenames or wildcard patterns, separate each filename with a plus (+) sign or a comma (,). You can specify the csource option as many times as necessary. See also the descriptions of the assembler , cxxsource , object , and library options.

1.37 cxxonly

CxxOnly

tells sc to translate the C++ source files into C source files without compiling them. The default value is nocxxonly.

The filename for the generated C file is

root-filename..c

root-filename is the filename of the C++ file. For example, if your C++ file is named foo.cxx, the C source file generated is named foo..c. If you do not specify the cxxonly option, the ..c file is created, compiled, and then deleted.

1.38 cxxsource

```
CxxSource=filename
```

specifies C++ source files that are to be compiled and, if you specify the link option, linked into the program. The minimum acceptable abbreviation is cxxsrc. This option does not have a negative form.

You can use AmigaDOS wildcard characters to specify filenames. To specify several filenames or wildcard patterns, separate each filename with a plus (+) sign or a comma (,). You can specify the cxxsource option as many times as necessary. See also the descriptions of the assembler , csource , object , and library options.

1.39 data

```
Data=reference-type
```

specifies whether you want the compiler to generate 16-bit or 32-bit references to external and static data items. You can specify any of the following:

near or n
  tells the compiler to use 16-bit references. If you specify near, all data not declared with the __far or __chip keyword are placed into the near data section. The default value is near.

far or f
  tells the compiler to use 32-bit references. Register A4 is still reserved to point to the near data section so that you can mix code compiled with data=near and data=far.

faronly or fo

  tells the compiler that your program never uses near data. If you specify faronly, the compiler generates 32-bit references and may use register A4 as an additional register variable. If you compile with the data=faronly option, and you declare data with the __near keyword, the compiler displays the warning message 194:

```
too much local data for NEAR reference,
some changed to FAR
```

  If your entire project is in one source file or is compiled with data=faronly, you can ignore this warning unless you get an error later in the compilation or link.

auto or a

  indicates that the first 32k of external data should generate 16-bit references and the remaining external data should generate 32-bit references. If your module has more than 32k of external

```
  data, the compiler displays the warning message 194:

      too much local data for NEAR reference,
      some changed to FAR


  If your entire project is in one source file or is compiled with
  data=faronly, you can ignore this warning unless you get an error
  later in the compilation or link.

  You should not use data=auto if you have multiple modules that
  share data.


This option does not have a negative form.

You can override this option on individual data items by using the
__near, __far, or __chip keywords.  __near forces the compiler to
generate a 16-bit reference, and __far forces the compiler to
generate a 32-bit reference.  __chip forces the compiler to place
the data item into chip memory.  For more information, refer to the
section "Using Special Keywords" in Chapter 11, "Using SAS/C
Extensions to the C and C++ Languages."

See also the description of the  code  option.
```

1.40 datamemory

```
DataMemory=type

specifies the type of memory into which initialized static or
external data should be loaded.  You can specify any, chip, or fast.
You can abbreviate these values as a, c, or f.  The default value is
any.  This option does not have a negative form.

This option affects code generated by both the compiler and the
assembler.  See also the descriptions of the  bssmem  and  codemem
options.
```

1.41 dataname

```
DataName=name

names the initialized data section.  The default value is data.  You
can specify dataname=none or nodataname if you want an unnamed data
section.

The linker automatically merges all sections with the same name.
See also the descriptions of the  bssname  and  codename  options.  See
Chapter 12, "How the Compiler Works" for information on data and
code sections.
```

1.42 debug

```
Debug=level

sets the debugging level of the compiler.  If you do not want the
compiler to generate debugging information, specify nodebug.  The
default value is nodebug.  The minimum acceptable abbreviation is
dbg.

To generate debugging information, specify debug=level, where level
is one of the following:

line or l
  produces line number information only.

symbol or s
  produces line number information, information on automatic and
  formal variables, and information on external and static symbols
  that are referenced in the module being compiled.

symbolflush or sf
  produces the same information as symbol, and flushes any
  non-register variables being held in registers to memory at each
  line boundary to allow the debugger to accurately display their
  values in C source mode.

full or f
  produces the same information as symbol.  However, debug=full
  produces information on all symbols whether or not the module
  references the symbol.

fullflush or ff
  produces the same information as full, and flushes any
  non-register variables being held in registers to memory at each
  line boundary to allow the debugger to accurately display their
  values in C source mode.


Any debug option except nodebug adds the -d assembler option to any
assembled files to force debugging line number information on
assembler output.  Also, if you specify the  link  option and any
debug option except nodebug, the  addsym  option is passed to the
linker.
```

1.43 define

```
Define[=]symbol[=value]

defines the specified preprocessor symbol to be used by the
compiler.  This option works as if you defined the symbol with a
#define statement. The minimum acceptable abbreviation is def.  This
option does not have a negative form.

Do not enter a space between the symbol name and the following equal
```

sign. If the value contains a space, enclose the entire argument in double quotes ("). As with all other compiler options, the equal sign between the define option and the symbol name is optional, so both of the following examples are acceptable:

```
define foo=bar
define=foo=bar
```

You can specify the define option as many times as necessary.

Any symbols defined with the define option are defined in the assembler as well.

NOTE: The define compiler option does not affect the linker. Do not confuse this option with the define linker option.

1.44 disassemble

DisAssemble=filename

tells the compiler to disassemble the code as it is generated and to send the disassembly to the file you specify. The default value is nodisassemble. The minimum acceptable abbreviation is disasm.

To send the disassembly to standard output, use disasm=*.

1.45 dollarok

DollarOK

allows dollar signs ($) as valid characters in identifiers in both C and C++ programs. The default value is nodollarok because the ANSI C Standard does not allow dollar signs in identifiers. The minimum acceptable abbreviation is dolok.

1.46 error

Error=n

tells the compiler to treat the specified message as an error. You can specify all or a to promote all enabled warnings to errors, or you can specify one or more message numbers to promote only those messages. The minimum acceptable abbreviation is err. This option does not have a negative form.

To specify several message numbers, separate each number with a plus (+) sign or a comma (,). You can specify the error option as many times as necessary. See also the descriptions of the warn and ignore options.

1.47 errorconsole

```
ErrorConsole
```

enables printing of diagnostics to the console (stdout). The default value is errconsole. The minimum acceptable abbreviation is errcon.

1.48 errorhighlight

```
ErrorHighlight
```

highlights the token that caused the error using ANSI escape sequences in diagnostic output that is sent to the console. The default value is errorhighlight. The minimum acceptable abbreviation is errhigh.

This option is ignored when compiling C++ files.

1.49 errorlist

```
ErrorList
```

prints diagnostic messages to the listing file. The default value is errorlist. The minimum acceptable abbreviation is errlist. This option is ignored if you do not specify the list option.

1.50 errorrexx

```
ErrorRexx
```

sends diagnostic messages from the compiler to the scmsg utility. The default value is noerrorrexx. The minimum acceptable abbreviation is errrexx.

For more information, refer to the description of the scmsg utility in Chapter 10, "Utility Reference," of SAS/C Development System User's Guide, Volume 2.

1.51 errorsource

```
ErrorSource
```

prints lines from the C source file with the diagnostic messages that are sent to the console. The default value is errorsource. The minimum acceptable abbreviation is errsrc.

1.52 externaldefs

```
ExternalDefs
```

treats all external definitions as definitions. The default value is externaldefs. The minimum acceptable abbreviation is extdef.

If you specify noexternaldefs, all external definitions are treated as external declarations. This action has the same effect as if you had declared each variable with the extern keyword. For example, int i is treated as extern int i, and it would need to be defined in a file compiled without noexternaldefs.

1.53 findsymbol

```
FindSymbol=symbol-name
```

tells the compiler to print a warning message each time the specified symbol is defined. The minimum abbreviation is fsym. This option does not have a negative form. You can specify the findsymbol option as many times as necessary.

If you specify the findsymbol option, a message is produced for any definition of the symbol, including #define statements, structure and union declarations, prototypes, and extern, static, and local variable definitions. Use the fsym option to quickly determine where a given preprocessor symbol or prototype is coming from.

This option is ignored when compiling C++ files.

1.54 from

```
From
```

is included only for compatibility with the slink command. The compiler ignores this option. This option does not have a negative form.

1.55 genprotodataitems

```
GenProtoDataItems
```

generates external declarations for variables defined in the source files that are not defined as static. The minimum abbreviation is gpdata. The default value is gpdata. This option is ignored if you do not specify the genprotos option.

1.56 genprotoexterns

GenProtoExterns

generates prototypes for externally-known routines. The default value is genprotoexterns. The minimum acceptable abbreviation is gpext.

This option is ignored if you do not specify the genprotos option.

1.57 genprotofile

GenProtoFile=filename

specifies the name of the file in which to place the generated prototypes. The default value is filename_protos.h. The minimum acceptable abbreviation is gpfile.

This option is ignored if you do not specify the genprotos option.

1.58 genprotoparameters

GenProtoParameters

generates prototypes using the __PARMS macro. The default value is nogenprotoparameters. The minimum acceptable abbreviation is gpparm.

This option allows your C code to compile successfully on compilers that support prototypes and on those that do not. On ANSI compilers, the __PARMS macro expands to the parameter list for the function, thereby creating a prototype. On non- ANSI compilers, the __PARMS macro expands to an open-close parentheses pair, which declares the function's return type without defining a prototype. This option is ignored if you do not specify the genprotos option.

1.59 genprotos

GenProtos

generates prototypes and data declarations instead of compiling your file. The default value is nogenprotos. The minimum acceptable abbreviation is gproto.

This option defines the preprocessor symbol _GENPROTO . If you specify a filename with the genprotofile option, the prototypes are written to the specified file. Otherwise, the prototypes are written to the file filename_protos.h.

While generating prototypes, the compiler suppresses most warnings automatically, because many of the warnings may be due to incorrect or missing prototypes. The compiler also checks all #include statements as they are reached. If your file #includes the same prototype file that is being generated, the compiler skips that #include statement. This feature allows you to use this option to maintain declarations for all externally-known symbols in each C source file and regenerate the declarations as the files change.

To set up your project so that you can use this option to maintain prototype files, do the following:

1. Create a header file that contains #include statements for each of the files in your project, as follows:

```
#include "file1_protos.h"
#include "file2_protos.h"
.
.
.
#include "filen_protos.h"
```

2. Include this header file in each file in your project.

3. Compile your entire project with the genproto option.

As each .c file is compiled, the compiler creates the corresponding _protos.h file. The compiler suppresses the header file not found warnings that would normally be produced.

This option is ignored when compiling C++ files.

1.60 genprotostatics

GenProtoStatics

generates prototypes for static routines. The default value is nogenprotostatics. The minimum acceptable abbreviation is gpstat.

This option is ignored if you do not specify the genprotos option.

1.61 genprototypedefs

GenProtoTypedefs

tells the compiler to use typedefs instead of resolved types when generating prototypes for any functions using typedefs for parameters or return values. The default value is genprototypedefs. The minimum acceptable abbreviation is gptdef.

This option is ignored if you do not specify the genprotos option.

1.62 globalsymboltable

```
GlobalSymbolTable=gst
```

tells the compiler to use the specified GST (Global Symbol Table). The default value is noglobalsymboltable. The minimum acceptable abbreviation is gst.

The GST must have been created using the makeglobalsymboltable option during a previous compilation. The gst option is ignored if you specify the makeglobalsymboltable option. Therefore, you can enter the makeglobalsymboltable in the sc command even if your scoptions file contains the gst option.

This option defines the preprocessor symbol _GST .

This option is ignored when compiling C++ files.

1.63 gst

```
GST=gst-filename
```

is a synonym for the GlobalSymbolTable option.

1.64 gstimmediate

```
GSTImmediate
```

is included for compatibility with projects using precompiled header files as implemented in Version 5 of the compiler. This option makes the contents of the GST you specify with the gst option immediately available to the program. The default value is nogstimmediate. The minimum acceptable abbreviation is gstimm.

Normally, symbols defined in a specific header file in the GST are available to your program only after you have included the header file with a #include statement. This option makes the contents of the GST you specify with the gst option immediately available to the program, even if your program does not contain #include statements for the header file. With Version 5 precompiled header files, all symbols in the precompiled header files were available to your program even if your program did not contain #include statements for the header file.

1.65 icons

```
Icons
```

tells the compiler to create icons for files that it creates,
including listing files, preprocessor output files, prototype files,
and object files. The default value is icon.

If you specify icons, then each time the compiler generates a file,
it looks in the drawer sc:icons for an icon named def_extension,
where extension is the filename extension of the file it created.
If the compiler finds an icon file appropriate to the file
extension, it copies the icon to the directory in which the file was
created. If the compiler cannot find sc:icons or cannot find an
icon with the appropriate extension, it does not create an icon. If
you specify noicons and link, the noicons option is also passed to
the linker. For more information, see the section "Using Icons" in
Chapter 2, "Using Your SAS/C Development System."

1.66 identifierlength

IdentifierLength=n

specifies the maximum number of significant characters in an
identifier. The default value is 31.

The minimum acceptable abbreviation is idlen. This option does not
have a negative form.

Identifiers longer than n are truncated without warning.
Identifiers longer than n bytes that differ after the first n bytes
are treated as identical.

This option is ignored when compiling C++ files.

1.67 ignore

Ignore=n

tells the compiler to ignore the specified warning message. The
minimum acceptable abbreviation is ign. This option does not have a
negative form.

You cannot ignore error messages. You can specify all or a to
ignore all warning messages, or you can specify one or more message
numbers to ignore only those messages. To specify several message
numbers, separate each number with a plus (+) sign or a comma (,).
You can specify the ignore option as many times as necessary.

See also the descriptions of the error and warn options.

1.68 includedirectory

IncludeDirectory=directory

adds a directory to the list of directories to search for include files. The default list is the current directory and include:. The minimum acceptable abbreviation is idir. This option does not have a negative form. You can specify the incdirectory option as many times as necessary.

Any directories you specify with includedirectory are also passed to the assembler as header file search directories.

1.69 keepline

KeepLine

generates #line directives in the preprocessor output file that correspond to the lines in the original source files. The default value is nokeepline. The minimum abbreviation is kline.

This option allows you to compile the preprocessed source and get error and warning messages that refer you to the correct line in the nonpreprocessed version of the file.

This option is ignored if you do not specify the pponly option.

1.70 libcode

LibCode

tells the compiler that the compiled code will be linked into a shared library. The default value is nolibcode.

Any functions compiled with libcode and either the __saveds keyword or the saveds option load their near data section from a point relative to the library base register A6 instead of from an absolute address.

libcode also guarantees that the current library base will be in register A6 whenever A6 is referenced or an internal call is made.

This option is for use when creating a shared library. Do not use this option when creating an object module for a normal executable.

1.71 libfd

LibFD=filename

specifies the name of a function description (.fd) file. The compiler passes this option to the linker. This option does not

have a negative form and is ignored if you do not specify the link option.

This option is for use when creating shared libraries. Do not use this option when creating an object module for a normal executable.

1.72 libprefix

LibPrefix=prefix

specifies the prefix that you want added to the functions listed in the function description (.fd) file. The compiler passes this option to the linker. This option does not have a negative form and is ignored if you do not specify the link option.

This option is for use when creating shared libraries. Do not use this option when creating an object module for a normal executable.

1.73 library

Library=link-library-filename(s)

specifies the link libraries that are to be passed to the linker. The minimum acceptable abbreviation is lib. This option does not have a negative form.

You can use AmigaDOS wildcard characters to specify filenames. To specify several filenames or wildcard patterns, separate each filename with a plus (+) sign or a comma (,). You can specify the library option as many times as necessary. Any libraries you specify are passed to the linker before the SAS/C libraries. This option is ignored if you do not specify the link option.

For more information, refer to the description of library in the linker options documentation.

See also the description of the csource , cxxsource , object , and asm options.

1.74 librevision

LibRevision=n

specifies a minor revision number for the shared library that you are creating. The compiler passes this option to the linker. This option does not have a negative form and is ignored if you do not specify the link option.

This option is for use when creating a shared library. Do not use this option when creating an object module for a normal executable.

1.75 libversion

LibVersion=n

sets the version number of the shared library that you are creating. The compiler passes this option to the linker. This option does not have a negative form and is ignored if you do not specify the link option.

This option is for use when creating a shared library. Do not use this option when creating an object module for a normal executable.

1.76 link

Link

tells the compiler to invoke the linker to produce a final executable module. The default value is nolink.

If you do not specify link, the compiler ignores any object files and link libraries that you specify.

The options passed to the linker are placed into the file program.lnk, and the linker is invoked using this file as a with file. To see which linker options were generated, look at the .lnk file after sc runs the linker.

1.77 linkerdefine

LinkerDefine[=] symbol[=value]

defines a symbol to be used in the linking process. You can specify as many LinkerDefine options as you like. The compiler passes each LinkerDefine option to the linker as the define option. The minimum acceptable abbreviation is linkdef. This option does not have a negative form and is ignored if you do not specify the link option.

1.78 linkeroptions

LinkerOptions=option(s)

passes the provided parameter to the linker's command line. The default value is nolinkeroptions. The minimum acceptable abbreviation is linkopt.

If you want to specify more than one option, or if the option you want to specify contains a blank, surround the entire option string

with double quotes ("), as in the following example:

```
   sc "linkeroptions=bufsize 10000 maxhunk 64000" link myprog.c
```

Any options specified in the options string are passed to the linker after any compiler options that are identified as valid only if you specify link. Therefore, the linkeroptions values override the values passed by the sc options.

This option is ignored if you do not specify the link option.

1.79 linkerwith

LinkerWith=filename

specifies the name of a file containing linker options. You can specify as many LinkerWith options as necessary. The compiler passes each LinkerWith option to the linker as the with option. The minimum acceptable abbreviation is linkwith. This option does not have a negative form and is ignored if you do not specify the link option.

1.80 list

List

tells the compiler or assembler to produce a listing file. The default value is nolist.

The compiler writes the output to the filename you specify with the listfile option. If you do not specify an output filename, the compiler uses the name of the first source file you specify but with the extension .lst.

This option is ignored when compiling C++ files.

1.81 listfile

ListFile=filename

names the listing and/or cross reference file. The default filename is the same filename as the source file but with the extension .lst. The minimum acceptable abbreviation is lfile.

This option is ignored if you do not specify the list or xreference options.

1.82 listheaders

```
ListHeaders

tells the compiler or assembler to include user header files in the
listing.  The default value is listheaders.  The minimum acceptable
abbreviation is lhead.

This option is ignored if you do not specify the list option.
```

1.83 listincludes

```
ListIncludes

lists the names of all included .h files in the listing file.  The
default value is listincludes.  The minimum acceptable abbreviation
is linc.

This option is useful for determining:

  - .h file dependencies for smake
  - the exact path of each .h file used
  - exactly which .h files are included by other .h files.

The hierarchy of files is indicated by indention levels; a file
included by another file is indented one level deeper than the
parent file.  This option is ignored if you do not specify the list
option.
```

1.84 listmacros

```
ListMacros

tells the compiler or assembler to expand macros in the listing.
The default value is nolistmacros.  The minimum acceptable
abbreviation is lmac.

This option is ignored if you do not specify the list option.
```

1.85 listnarrow

```
ListNarrow

tells the compiler or assembler to produce a narrow listing (less
than 80 columns wide).  The default value is listnarrow.  The
minimum acceptable abbreviation is lnarr.

This option is ignored if you do not specify the list option.
```

1.86 listsystem

```
ListSystem

tells the compiler to include system header files in the listing.
The default value is nolistsystem.  The minimum acceptable
abbreviation is lsys.

This option is ignored if you do not specify the list option.
```

1.87 makeglobalsymboltable

```
MakeGlobalSymbolTable=gst-filename

creates a GST (Global Symbol Table).  The minimum acceptable
abbreviation is mgst.

If you specify the  gst  and makegst options, the gst option is
ignored.  Therefore, you can enter the makegst in the sc command
even if your scoptions file contains the gst option.  This option
automatically enables the  nomultipleincludes  and  noexternaldefs
options.  For more information on creating and using GSTs, refer to
SAS/C Development System Library Reference.

This option defines the preprocessor symbol  _MGST .

This option is ignored when compiling C++ files.
```

1.88 map

```
Map

produces a map of the executable module.  The default value is
nomap.  This option is ignored if you do not specify the  link
option.

For more information, refer to the description of  map
in the linker options documentation.
```

1.89 mapfile

```
MapFile=filename

names the map file.  The default value is executable.map.  The
minimum acceptable abbreviation is mfile.

This option is ignored if you do not specify the  link  and  map
options.
```

1.90 maphunk

MapHunk

maps all output hunks by size and originating function. The default value is maphunk. The minimum acceptable abbreviation is mhunk.

This option is ignored if you do not specify the link and map options.

1.91 maplib

MapLib

generates a list of hunks by library symbol in the map. The default value is nomaplib. The minimum acceptable abbreviation is mlib.

This option is ignored if you do not specify the link and map options.

1.92 mapoverlay

MapOverlay

includes a list of hunks in each overlay in the map. The default value is nomapoverlay. The minimum acceptable abbreviation is movly.

This option is ignored if you do not specify the link and map options.

1.93 mapsymbols

MapSymbols

includes a list of defined symbols and the location at which they are defined. The default value is nomapsymbols. The minimum acceptable abbreviation is msym.

This option is ignored if you do not specify the link and map options.

1.94 mapxreference

MapXreference

writes a symbol cross-reference to the map file that lists each symbol definition and the places each symbol is used. The default value is nomapxreference. The minimum acceptable abbreviation is mxref.

This option can generate a lot of output, but it is useful when you are trying to track down where an unresolved symbol is referenced. This option is ignored if you do not specify the link and map options.

1.95 math

Math=type

chooses a format for floating-point math and, if you also specify the link option, links with the appropriate math library. The default value is nomath.

You can specify one of the following as the type:

standard or s
   links with the library scm.lib, scms.lib or scmnb.lib, depending on the value of the data and shortint options. The math format is IEEE.

ffp or f
   generates code to call the FFP shared library provided by Commodore. The math format is FFP. This option links with scmffp.lib. This option is not supported for C++ files.

68881, 68882, or 8
   generates inline code for the 68881 and 68882 coprocessors. If you specify one of these coprocessor options, your program will not run if a coprocessor is not available. coprocessor is a synonym for these options. The math format is IEEE. This option links with scm881.lib.

ieee or i
   generates code to call the IEEE shared library provided by Commodore. The math format is IEEE. This option links with scmieee.lib.

Some math options define preprocessor symbols. See the section "Using Preprocessor Symbols Defined by the Compiler," later in this chapter, for a list of those symbols.

For additional information about compiling and linking with math libraries, refer to Chapter 3, "Using the SAS/C Libraries," in SAS/C Development System Library Reference.

1.96 maximumerrors

```
MaximumErrors=n
```

sets the limit on the number of errors for a single compilation. The default value is 50. The minimum acceptable abbreviation is maxerr.

If a single compilation generates more than n errors, the compiler aborts the compilation. nomaxerr removes any limits.

1.97 maximumwarnings

```
MaximumWarnings=n
```

sets the limit on the number of warnings for a single compilation. The default value is nomaxwrn. The minimum acceptable abbreviation is maxwrn.

If a single compilation generates more than n warnings, the compiler aborts the compilation. nomaxwrn removes any limits.

1.98 memorysize

```
MemorySize=size
```

tells the compiler approximately how much memory you have on your system. You can specify one of the following:

- tiny or t
- small or s
- medium or m
- large or l
- huge or h

The default value is large. The minimum acceptable abbreviation is memsize. This option does not have a negative form.

Larger sizes allow sc to compile more complex programs and to compile faster. Smaller sizes allow sc to continue to work under low-memory conditions. If the compiler runs out of memory during a compilation, it displays the message ***Freeing Resources, attempts to free up memory, and automatically drops to a lower memorysize value.

If you specify link and either memorysize=tiny or memorysize=small, the compiler passes the bufsize=4096 option to the linker.

memorysize affects how and where the compiler stores any debugging information. If the compiler begins to run out of memory, it starts writing debugging information to a disk file. This file is referred to as a debug side file.

memorysize also affects the disposition and buffering of the compiler intermediate file. This option tells the compiler how much initial memory space to reserve for the intermediate information. For large and huge, the intermediate file is kept totally in memory (when the memory is available), which is much faster than writing it to disk. At smaller values, the intermediate file is written to disk, but the memorysize value affects the amount that is buffered.

In addition, if you do not specify the preprocessorbuffer (ppbuf) and/or argumentsize (argsize) options, the memorysize option sets these values for you. If you specify the preprocessorbuffer and/or argumentsize options, the values you specify override the values set by memorysize.

The following table lists the default values for argumentsize and preprocessorbuffer by memorysize. It also lists the buffer size of the compiler intermediate file.

	Intermediate	Debug Side	
memsize	argsize	ppbuf	File Buffer	File Buffer
tiny	127	1024	1024	2K
small	255	2048	4096	8K
medium	511	4096	8192	32K
large	1023	8192	no limit	64K
huge	4800	16384	no limit	128K

See also the descriptions of the preprocessorbuffer and argumentsiz options.

1.99 modified

Modified

tells the compiler to process only files that are out of date with respect to their output files. The default value is nomodified. The minimum acceptable abbreviation is mod.

This option is useful if you are compiling several files with a single sc command and only some of the files need to be recompiled. This option also works when you are generating prototypes or preprocessor output. If you also specify the link option, all object files are included in the link, even if their source files were not recompiled.

1.100 multiplecharacterconstants

MultipleCharacterConstants

allows up to four bytes to appear within single quotes as a

character constant. This option is included only for compatibility with previous releases of the compiler, and its use is not recommended. The minimum acceptable abbreviation is mccons. The default value is nomccons.

If you specify mccons, a single constant of type int is generated. If fewer than four bytes are provided, they are padded on the left with zeroes, as in the following example:

```
#include <stdio.h>

void main(void)
{
   long l = 'abcd';
   long m = '\x01\x02\x03';
   printf("l=0x%08lx, m=0x%08lx\n", l, m);
}
```

This example program prints the following:

```
l=0x61626364, m=0x00010203
```

This option is ignored when compiling C++ files.

1.101 multipleincludes

MultipleIncludes

tells the compiler to include header files that are included more than once. The default value is multipleincludes. The minimum acceptable abbreviation is minc.

This behavior is required by the ANSI Standard. However, many projects do not need this behavior, and specifying nominc can save compilation time.

Specifying nomultipleincludes has no effect when compiling C++ files.

1.102 object

Object=filename(s)

lists the object files that are to be linked into the program. The minimum acceptable abbreviation is obj. This option does not have a negative form.

The object files must have been created during a previous compilation. Do not specify object files that will be created by the same sc command in which you specify the object option. You can

use AmigaDOS wildcard characters to specify filenames. To specify several filenames or wildcard patterns, separate each filename with a plus (+) sign or a comma (,). You can specify the object option as many times as necessary.

This option is ignored if you do not specify the link option. See also the descriptions of the csource , library , and assembler options.

1.103 objectlibrary

ObjectLibrary=link-library-name

specifies that any resulting object files are to be placed in the named link library. The minimum acceptable abbreviation is objlib.

Do not specify this option if you also specify the link option.

1.104 objectname

ObjectName=file-or-directory-name

specifies the name of the file or directory to hold compiler and assembler output (including temporary ..c files). The minimum acceptable abbreviation is objname.

If you are compiling or assembling only one source file, you can use this option to specify the file where you want the compiler or assembler output. If you are compiling or assembling more than one file, you can use this option to specify a directory where you want the output, and sc uses the same filename as the source file but replaces the extension with .o. The directory name must end with a forward slash (/) or a colon (:). If you do not use the objectname option, sc uses the same filename with the extension of .o, but it places the files in the same directory as the source file.

The C++ translator will generate the temporary .c file into the same path as specified by the ObjectName option.

1.105 oldpreprocessor

OldPreprocessor

is provided for compatibility with pre- ANSI style preprocessors. The default value is nooldpreprocessor. The minimum acceptable abbreviation is oldpp.

The oldpp option does the following:

- allows old-style token pasting using comments

- substitutes values for parameters specified as quoted strings in macro definitions.

For example, suppose you have the following program:

```
#define FOO(bar) "bar"

void main(void)
{
   printf("%s\n", FOO(test));
}
```

If you compile this program with oldpp, the program prints test. If you compile this program with nooldpp, the program prints bar. To make this program work with the ANSI features and the nooldpp option, substitute the following for the #define:

```
#define FOO(bar) #bar
```

As an additional example, suppose you have the following program:

```
#define FOO(bar) foo_/**/bar

void main(void)
{
   int foo_test;
   int foo_xxx;

   FOO(test) = 10;
   FOO(xxx)  = 20;
   printf("%d %d\n", foo_test, foo_xxx);
}
```

If you compile this program with oldpp, the #define concatenates the text of the argument after the string foo_ and the program prints 10 20. If you compile this program with nooldpp, the program produces a compilation error. To make this program work with the ANSI features and the nooldpp option, substitute the following for the #define:

```
#define FOO(bar) foo_##bar
```

This option is ignored when compiling C++ files.

1.106 onerror

OnError=x

tells sc what action to take if a source file generates an error. The minimum acceptable abbreviation is onerr. This option does not have a negative form.

You can specify one of the following:

```
stop or s
   tells sc not to process any more source files.  The default value
   is stop.

continue or c
   tells sc to process the next source file.

If you specify the  link  option, but your program generates errors,
your program is not linked even if you specify onerr=continue.
```

1.107 optimize

```
Optimize

enables the global optimizer (unless you specify nooptglobal) and
peephole optimizer (unless you specify nooptpeep).  The default
value is nooptimize.  The minimum acceptable abbreviation is opt.
```

1.108 optimizeralias

```
OptimizerAlias

disables type-based aliasing assumptions in the optimizer.  The
default value is nooptimizeralias.  The minimum acceptable
abbreviation is optalias.

If you specify optimizeralias, the global optimizer uses worst-case
aliasing.  Specifying optimizeralias can significantly reduce the
amount of optimization that can be performed.  This option is
ignored if you do not specify the  optimize  and  optglobal  options.
```

1.109 optimizercomplexity

```
OptimizerComplexity=n

defines the maximum complexity level of functions to be
automatically inlined.  The default value is 0.  The minimum
acceptable abbreviation is optcomp.

The parameter n represents the relative complexity of the function
to be inlined and is a count of the number of discrete operations in
the function.  Try different values for this number until you get
the results you want.  The higher the number, the more functions you
can inline, but the size of your code will grow significantly as
well.

If you specify nooptcomp, no complexity-based inlining occurs.  This
option is ignored if you do not specify the  optimize ,  optglobal , and
 optimizerinline  options.
```

1.110 optimizerdepth

```
OptimizerDepth=n
```

defines the maximum nesting depth of automatically inlined functions. The default value is 0. The minimum acceptable abbreviation is optdep.

This option is ignored if you do not specify the optimize , optglobal , and optimizerinline .

1.111 optimizerglobal

```
OptimizerGlobal
```

enables the global optimizer. The default value is optimizerglobal. The minimum abbreviation is optgo. This option is ignored if you do not specify the optimize option.

1.112 optimizerinline

```
OptimizerInline
```

allows inlining of functions, including functions defined with the __inline keyword. The default value is optimizerinline. The minimum acceptable abbreviation is optinl.

This option is ignored if you do not specify the optimize and optglobal options.

If you specify nooptinline, the optinlocal , optdepth , optcomplexity , and optrdepth options are ignored.

1.113 optimizerinlocal

```
OptimizerInLocal
```

inlines single-use static functions. The default value is nooptimizerinlocal. The minimum acceptable abbreviation is optinlocal.

This option is ignored if you do not specify the optimize , optglobal , and nooptimizerinline options.

1.114 optimizerloop

```
OptimizerLoop

enables loop optimizations.  The default value is optimizerloop.
The minimum acceptable abbreviation is optloop.

This option is ignored if you do not specify the  optimize  and
 optglobal  options.
```

1.115 optimizerpeephole

```
OptimizerPeephole

enables the peephole optimizer.  The default value is
optimizerpeephole.  The minimum acceptable abbreviation is optpeep.

This option is ignored if you do not specify the  optimize  option.
```

1.116 optimizerrecurdepth

```
OptimizerRecurDepth=n

defines the maximum depth of recursion of automatically inlined
functions.  The default value is 0.  The minimum acceptable
abbreviation is optrdep.

This option is ignored if you do not specify the  optimize ,
 optglobal , and  optinline  options.
```

1.117 optimizersize

```
OptimizerSize

generates smaller code, possibly increasing execution time.  The
default value is nooptsize.  The minimum acceptable abbreviation is
optsize.

This option is ignored if you do not specify the  optimize  and
 optglobal  options.  Do not specify both the optimizersize and
 optimizertime  options.
```

1.118 optimizertime

OptimizerTime

generates code that will run faster but may be larger. The default value is nooptimizertime. The minimum acceptable abbreviation is opttime.

This option is ignored if you do not specify the optimize and optglobal options. Do not specify both the optimizersize and optimizertime options.

1.119 optimizerschedule

OptimizerSchedule

runs the instruction scheduler. The default value is nooptschedule. The minimum acceptable abbreviation is optsched.

The scheduler reorders instructions to make them run faster on higher-order processors, like the 68040 and 68882, without sacrificing the ability to run on lower-order processors. Running the scheduler may take additional compilation time.

This option is ignored if you do not specify the optpeep and optimize options.

1.120 parameters

Parameters=method

indicates how parameters should be passed. The minimum acceptable abbreviation is parm. This option does not have a negative form.

You can specify one of the following:

stack or s
  indicates parameters should be passed on the stack. The default value is stack.

register or r
  indicates parameters should be passed in registers.

both or b
  generates a combination prolog that allows parameters to be passed on the stack or in registers.

If your program has prototypes for all routines, you should probably use parameters=register for increased efficiency. If you are placing code into a link library, specify parms=both so that your library functions accept registerized parameters and parameters that are passed on the stack.

If you write a function that has the same name as a SAS/C library function, you need to add the __regargs keyword to your function or compile with the parms=both or parms=register option. For example, you may want to replace the SAS/C library function malloc with your own version of malloc. If you compile with the parms=stack option or define your version of malloc with the __stdargs keyword, then two versions of malloc are linked into your executable. If you use other SAS/C library functions that call malloc, these functions use the version of malloc in the SAS/C libraries. However, your functions that call malloc use your version of malloc. To make sure that all calls to malloc are using your version of malloc (including calls from library routines), define your version with __regargs or compile with the parms=both or parms=register option.

With C++ functions, only functions with "C" linkage obey the value you specify with the parameters option. All C++ functions behave as if you specified parameters=register unless you explicitly declare them with __stdargs or __asm.

For information about passing parameters between C language functions and assembly language functions, refer to Chapter 11, "Using Assembly Language with the C and C++ Languages," in SAS/C Development System User's Guide, Volume 2.

1.121 precision

Precision=precision

specifies the size of floating-point variables. You can specify double or mixed. The default value is mixed. The minimum acceptable abbreviation is prec. This option does not have a negative form.

If you specify double, data declared as float are treated as if they were declared as double. If you specify mixed, data declared as float and data declared as double are different sizes.

1.122 preprocessorbuffer

PreprocessorBuffer=n

sets the maximum number of bytes to which a macro can be expanded by the preprocessor. The default value is 8192 unless set differently by the memorysize option. The minimum acceptable abbreviation is ppbuf. This option does not have a negative form.

If a macro expands to greater than n bytes, the compiler issues an error message and aborts the compilation. See the description of the memorysize option for more information.

This option is ignored when compiling C++ files.

1.123 preprocessoronly

```
PreprocessorOnly

tells the compiler to run only the preprocessor on the source files.
The default value is nopreprocessoronly.  The minimum acceptable
abbreviation is pponly.

If you specify a filename with the  objectname  option, the compiler
writes the output to the file you specify.  If you do not specify an
output filename, the compiler writes the output to the same filename
as the source file but with the extension .p.

This option defines the preprocessor symbol  _PPONLY .
```

1.124 profile

```
Profile

generates code at the entry and exit of each function that calls
library functions _PROLOG and _EPILOG, respectively.  _PROLOG and
_EPILOG note the time the function was entered and exited and pass
this information to sprof, which produces a report telling you how
much time was spent in each function. The default value is
noprofile.  This option defines the preprocessor symbol  _PROFILE .
```

1.125 programname

```
ProgramName=output-module-name

specifies the name of the executable module.  The default value is
the root name of the first source file specified in the sc command.
The minimum acceptable abbreviation is pname.  This option does not
have a negative form.

This option is ignored if you do not specify the  link  option.
```

1.126 pubscreen

```
PubScreen=name

specifies the name of the public screen to use when sc invokes
scmsg.  The minimum abbreviation is pubscr.  This option does not
```

have a negative form.

The pubscreen option overrides any public screen specification in the scmsg options file. The scopts utility uses any pubscreen specification set in the scoptions file or on the command line.

1.127 resetoptions

ResetOptions

resets all options to their default values. The minimum acceptable abbreviation is resopt. This option does not have a negative form.

If you specify this option as the first option in the sc command, it resets any option specified in the scoptions file. If you specify this option after other options or filenames in the sc command, it resets the options preceding it, including the filenames.

1.128 saveds

Saveds

generates code as if you had defined all functions in the source files with the __saveds keyword. The default value is nosaveds.

The __saveds keyword loads the near data pointer in register A4 at each function entry point. You should specify this option if you are compiling code that is used as an interrupt routine, called with the AddTask function, used in a shared library, or called from another process. This option does not work if your code is linked with the cres.o or catchres.o startup modules.

1.129 shortintegers

ShortIntegers

enables 16-bit integers. The default value is noshortintegers. The minimum acceptable abbreviation is sint.

If you specify shortintegers, types int and short are the same size. If you specify noshortintegers, types int and long are the same size. If you are running the assembler, this option defines the symbol shortint. This option also defines the preprocessor symbol _SHORTINT .

This option is illegal when compiling C++ files.

1.130 smallcode

```
SmallCode

tells the linker to merge all code hunks into a single hunk.  The
default value is nosmallcode.  The minimum acceptable abbreviation
is scode.

This option is ignored if you do not specify the  link  option.  See
also the description of the  smallcode  linker option.
```

1.131 smalldata

```
SmallData

tells the linker to merge all data and bss sections into a single
hunk.  The default value is nosmalldata.  The minimum acceptable
abbreviation is sdata.

This option is ignored if you do not specify the  link  option.  See
also the description of the  smalldata  linker option.
```

1.132 sourceis

```
SourceIs=filename

sets the name of the C source file in the object file and in
debugging information to the specified value instead of the actual
name of the C source file.  The minimum acceptable abbreviation is
srcis.

You can use this option if you plan to rename or move the source
file before using the debugger to debug your program.  If you are
compiling or assembling exactly one source file, sourceis can
specify the name of the file to be placed into the debug
information.  If you are compiling or assembling multiple files,
sourceis should specify a directory name (ending with a forward
slash or colon).
```

1.133 stackcheck

```
StackCheck

generates stack overflow checking code at each function entry.  The
default value is stackcheck.  The minimum acceptable abbreviation is
stkchk.

When a program is about to run out of stack space, the program
displays a requestor and terminates gracefully.  For a complete
```

description of the stackcheck option, see Chapter 11, "Using SAS/C Extensions to the C and C++ Languages."

1.134 stackextend

StackExtend

generates stack extension code at each function entry. The default value is nostackext. The minimum acceptable abbreviation is stkext.

When a program runs out of space in the current stack, a new stack is allocated, and your program continues to run. For a complete description of the stackext option, see Chapter 11, "Using SAS/C Extensions to the C and C++ Languages."

1.135 standardio

StandardIO

is included for compatibility with previous releases of the compiler. This option has no effect. The libraries in Version 6.50 and later automatically initialize standard I/O if it is used.

1.136 startup

StartUp=module-name

specifies which startup module to use. The default value is c (for c.o). The minimum acceptable abbreviation is strt.

If the module name that you specify does not contain a colon (:), forward slash (/), or period (.), the compiler adds lib: to the beginning and .o to the end of the name you specify. If you do not want to compile with a startup module, specify nostartup. This option is ignored if you do not specify the link option.

If you specify libinit or libinitr, then the compiler also uses LIB:libent.o before the specified startup file and tells the linker to generate a shared library. If you specify devinit or devinitr, then the compiler also uses LIB:devent.o before the specified startup file and tells the linker to generate a device.

1.137 strict

Strict

enables a large number of diagnostics dealing with portability and questionable situations in your code. The default value is nostrict.

Specifying strict may produce warning messages for situations that are not a problem.

For more information on improving the portability of your code, see Chapter 13, "Writing Portable C Code."

This option is ignored when compiling C++ files.

1.138 stringsconst

StringsConst

tells the compiler to consider all string constants to be of type const char *. The default value is nostringsconst. The minimum acceptable abbreviation is strcons.

If you do not specify stringsconst, the compiler considers string constants to be of type char *. If you specify stringsconst, passing a string constant to a function that expects the type char * generates a warning message indicating that the function may modify the string constant.

This option is ignored when compiling C++ files.

1.139 stringmerge

StringMerge

merges all identical string constants in the C source file and changes the default value of the stringsection option to stringsection=code. The default value is nostringmerge. The minimum acceptable abbreviation is strmer.

If you specify stringmerge, the compiler examines each string constant defined in the code and checks for duplicates. If the compiler finds a duplicate string constant, it forces both references to refer to the same memory location.

If you specify stringmerge and you modify a string constant, the constant is modified in all locations. For example, suppose you have the following program:

```
#include <stdio.h>

void modifyme(char *);
```

```
   void main(void)
   {
      modifyme("Hello, World!\n");
      printf("Hello, World!\n");
   }

   void modifyme(char *msg)
   {
      strcpy(msg, "Foobar\n");
   }
```

If you compile the program with stringmerge, the program prints Foobar. If you compile the program with nostringmerge, the program prints Hello, World!.

1.140 stripdebug

StripDebug

strips all debugging information from the final executable. The default value is nostripdebug. The minimum acceptable abbreviation is stripdbg.

This option is ignored if you do not specify the link option.

For more information, refer to the description of stripdebug in the linker options documentation.

1.141 stringsection

StringSection=section

indicates where to place strings, data declared static const, and initializers for automatic structures, unions, and arrays.

The minimum acceptable abbreviation is strsect. This option does not have a negative form.

You can specify one of the following:

default
   stores data according to the following rules:

   1. If you specify strmerge, the code section is used.
   2. If you specify data=far or data=near, the named section is used.
   3. Otherwise, the near section is used.

   The default value is default.

near or n

stores the data in the near data section.

far or f
stores the data in the far data section.

code or c
stores the data in the code section.

If you specify a value other than default, the strings and static const data are placed in the specified section regardless of whether you specify strmerge or data.

This option allows you to merge strings and place them in the far section for residentable programs or shared libraries. Since the strings are read-only, they can be placed safely into far data; placing them in the far section frees up more near data space for your read/write external variables.

The stringsection option has two useful side effects. Because some data are placed in the code section:

- Less data needs to be placed into the near data section. If you have more than 64k of data, you can use the stringsection option to try to reduce the amount of data and allow your code to continue to use the near data model.

- The string constants are addressed relative to the PC (program counter) instead of the beginning of the near data section. Therefore, it is possible to generate programs with no near or far data.

For more information on the code and near data sections, see Chapter 12, "How the Compiler Works"

1.142 structureequivalence

StructureEquivalence

tells the compiler not to issue messages if a pointer to one structure type is passed to a function when the function expects a pointer to a different type, if the type passed is equivalent to the type expected. The default value is nostructureequivalence. The minimum acceptable abbreviation is streq.

For information on using equivalent structures, see Chapter 11, "Using SAS/C Extensions to the C and C++ Languages." This option is ignored when compiling C++ files.

1.143 to

To=filename

is included only for compatibility with the slink command. to is a synonym for the programname option. This option does not have a negative form.

1.144 trigraph

Trigraph

specifies whether to use ANSI trigraphs in your programs. The default value is notrigraph. The minimum acceptable abbreviation is trig. Specifying trigraph slows down the compiler.

1.145 underscore

Underscore

adds underscores to the beginning of all external names defined in any assembler source files assembled. The default value is nounderscore. The minimum acceptable abbreviation is uscore.

The underscores allow you to refer to these names in assembly language in the same way you do in C and C++ source files. This option is ignored if you do not specify any assembly-language files in the sc command. For more information, refer to Chapter 11, "Using Assembly Language with the C and C++ Languages," in SAS/C Development System User's Guide, Volume 2.

1.146 unsignedchar

UnsignedChar

makes the default type of char variables unsigned instead of signed. The default value is nounsignedchar. The minimum acceptable abbreviation is uchar. This option defines the preprocessor symbol _UNSCHAR .

1.147 utilitylibrary

UtilityLibrary

generates inline calls to the AmigaDOS 2.0 ROM-resident library utility.library to do integer multiplication and division instead of calling SAS/C library functions to do these operations. The default value is noutilitylibrary. The minimum acceptable abbreviation is

utillib.

Specifying utilitylibrary makes your executable smaller and faster by taking advantage of 68020 instructions if they are available, but your program runs only under AmigaDOS 2.0. If you link using the sc command and the utilitylibrary and link options, sc defines all the SAS/C library integer conversion stub routines to stubs that call utility.library. This action prevents the SAS/C library routines that use integer conversion routines from using the SAS/C versions of the routines. If you link using the slink command, you need to specify the following define linker options:

```
define __CXM33=__UCXM33
define __CXD33=__UCXD33
define __CXM22=__UCXM22
define __CXD22=__UCXD22
```

These define options are in the file LIB:utillib.with, so you can link with the following command:

```
slink with LIB:utillib.with options
```

1.148 verbose

Verbose

displays messages about each stage of compiling and linking. The default value is noverbose.

1.149 version

Version

prints a banner containing the compiler version number and a copyright message. The default value is version. The minimum acceptable abbreviation is ver.

1.150 warn

Warn=n

enables the specified compiler warning message. The minimum acceptable abbreviation is wrn. This option does not have a negative form.

You can specify all or a to enable all warning messages, or you can specify one or more message numbers to enable only those messages. To specify several message numbers, separate each number with a plus (+) sign or a comma (,). You can specify the warn option as many

times as necessary.

See also the description of the error and ignore options.

1.151 warnvoidreturn

WarnVoidReturn

issues a warning message if a function declared as returning an integer actually returns no value. The default value is warnvoidreturn. The minimum acceptable abbreviation is wvret.

The nowarnvoidreturn option suppresses the return value mismatch warning message for functions declared as returning an integer, but that do not contain a return statement or that do not include an expression in the return statement.

This option is ignored when compiling C++ files.

1.152 with

With=filename

specifies the name of a file containing additional options. The additional options are read immediately, as if they were specified in the sc command at the position occupied by the with option. Options specified after the with option may override options specified in the with file. This option does not have a negative form. You can specify the with option as many times as necessary.

NOTE: Do not confuse this option with the with linker option.

1.153 xref

XREF

is a synonym for the xreference option.

1.154 xreference

XReference

produces a cross-reference. The default value is noxreference. The minimum acceptable abbreviation is xref.

The compiler writes the output to the filename you specify with the listfile option. If you do not specify an output filename, the

compiler uses the name of the first source file you specify but with
the extension .lst.

This option is ignored when compiling C++ files.

1.155 xreferenceheaders

XReferenceHeaders

generates a cross-reference of user header files. The default value
is xreferenceheaders. The minimum acceptable abbreviation is xhead.

This option is ignored if you do not specify the xreference option.

1.156 xreferencesystem

XReferenceSystem

includes symbols defined in system header files in the
cross-reference. The default value is xreferencesystem. The
minimum acceptable abbreviation is xsys.

This option is ignored if you do not specify the xreference option.

1.157 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.

1.158 addsym

AddSymbols

tells the linker to add symbol information to the executable module. The default value is noaddsymbols. The minimum acceptable abbreviation is addsym.

This option is automatically enabled if you specify the debug option. This option is ignored if you do not specify the link option.

For more information, refer to the description of AddSymbols in the linker options documentation.

1.159 chkabort

CheckAbort

enables Control-C in your program. The default value is checkabort. The minimum acceptable abbreviation is chkabort. This option is ignored if you do not specify the link option.

If you compile with checkabort, your program will check for Control-C interrupts whenever I/O is performed. If you specify nocheckabort, the compiler passes the following option to the linker:

```
define @__chkabort=@__dummy
```

dummy is a function provided in the link libraries that simply returns. Specifying nocheckabort makes all calls to the __chkabort function do nothing, thereby removing Control-C checking.

Do not specify nocheckabort if you are redefining the library function __chkabort in your own code.

1.160 datamem

DataMemory=type

specifies the type of memory into which initialized static or external data should be loaded. You can specify any, chip, or fast. You can abbreviate these values as a, c, or f. The default value is any. This option does not have a negative form.

This option affects code generated by both the compiler and the

assembler. See also the descriptions of the bssmem and codemem options.

1.161 disassem

DisAssemble=filename

tells the compiler to disassemble the code as it is generated and to send the disassembly to the file you specify. The default value is nodisassemble. The minimum acceptable abbreviation is disasm.

To send the disassembly to standard output, use disasm=*.

1.162 errorlisting

ErrorList

prints diagnostic messages to the listing file. The default value is errorlist. The minimum acceptable abbreviation is errlist. This option is ignored if you do not specify the list option.

1.163 genprotodataitem

GenProtoDataItems

generates external declarations for variables defined in the source files that are not defined as static. The minimum abbreviation is gpdata. The default value is gpdata. This option is ignored if you do not specify the genprotos option.

1.164 genprotoextern

GenProtoExterns

generates prototypes for externally-known routines. The default value is genprotoexterns. The minimum acceptable abbreviation is gpext.

This option is ignored if you do not specify the genprotos option.

1.165 genprotoparm

GenProtoParameters

generates prototypes using the __PARMS macro. The default value is nogenprotoparameters. The minimum acceptable abbreviation is gpparm.

This option allows your C code to compile successfully on compilers that support prototypes and on those that do not. On ANSI compilers, the __PARMS macro expands to the parameter list for the function, thereby creating a prototype. On non- ANSI compilers, the __PARMS macro expands to an open-close parentheses pair, which declares the function's return type without defining a prototype. This option is ignored if you do not specify the genprotos option.

1.166 genproto

GenProtos

generates prototypes and data declarations instead of compiling your file. The default value is nogenprotos. The minimum acceptable abbreviation is gproto.

This option defines the preprocessor symbol _GENPROTO . If you specify a filename with the genprotofile option, the prototypes are written to the specified file. Otherwise, the prototypes are written to the file filename_protos.h.

While generating prototypes, the compiler suppresses most warnings automatically, because many of the warnings may be due to incorrect or missing prototypes. The compiler also checks all #include statements as they are reached. If your file #includes the same prototype file that is being generated, the compiler skips that #include statement. This feature allows you to use this option to maintain declarations for all externally-known symbols in each C source file and regenerate the declarations as the files change.

To set up your project so that you can use this option to maintain prototype files, do the following:

1. Create a header file that contains #include statements for each of the files in your project, as follows:

```
#include "file1_protos.h"
#include "file2_protos.h"
.
.
.
#include "filen_protos.h"
```

2. Include this header file in each file in your project.

3. Compile your entire project with the genproto option.

As each .c file is compiled, the compiler creates the corresponding

_protos.h file. The compiler suppresses the header file not found warnings that would normally be produced.

This option is ignored when compiling C++ files.

1.167 genprotostatic

GenProtoStatics

generates prototypes for static routines. The default value is nogenprotostatics. The minimum acceptable abbreviation is gpstat.

This option is ignored if you do not specify the genprotos option.

1.168 genprototypedef

GenProtoTypedefs

tells the compiler to use typedefs instead of resolved types when generating prototypes for any functions using typedefs for parameters or return values. The default value is genprototypedefs. The minimum acceptable abbreviation is gptdef.

This option is ignored if you do not specify the genprotos option.

1.169 libraries

Library=link-library-filename(s)

specifies the link libraries that are to be passed to the linker. The minimum acceptable abbreviation is lib. This option does not have a negative form.

You can use AmigaDOS wildcard characters to specify filenames. To specify several filenames or wildcard patterns, separate each filename with a plus (+) sign or a comma (,). You can specify the library option as many times as necessary. Any libraries you specify are passed to the linker before the SAS/C libraries. This option is ignored if you do not specify the link option.

For more information, refer to the description of library in the linker options documentation.

See also the description of the csource , cxxsource , object , and asm options.

1.170 librev

```
LibRevision=n
```

specifies a minor revision number for the shared library that you are creating. The compiler passes this option to the linker. This option does not have a negative form and is ignored if you do not specify the link option.

This option is for use when creating a shared library. Do not use this option when creating an object module for a normal executable.

1.171 libver

```
LibVersion=n
```

sets the version number of the shared library that you are creating. The compiler passes this option to the linker. This option does not have a negative form and is ignored if you do not specify the link option.

This option is for use when creating a shared library. Do not use this option when creating an object module for a normal executable.

1.172 linkeropts

```
LinkerOptions=option(s)
```

passes the provided parameter to the linker's command line. The default value is nolinkeroptions. The minimum acceptable abbreviation is linkopt.

If you want to specify more than one option, or if the option you want to specify contains a blank, surround the entire option string with double quotes ("), as in the following example:

```
   sc "linkeroptions=bufsize 10000 maxhunk 64000" link myprog.c
```

Any options specified in the options string are passed to the linker after any compiler options that are identified as valid only if you specify link. Therefore, the linkeroptions values override the values passed by the sc options.

This option is ignored if you do not specify the link option.

1.173 makegst

```
MakeGlobalSymbolTable=gst-filename
```

creates a GST (Global Symbol Table). The minimum acceptable abbreviation is mgst.

If you specify the gst and makegst options, the gst option is ignored. Therefore, you can enter the makegst in the sc command even if your scoptions file contains the gst option. This option automatically enables the nomultipleincludes and noexternaldefs options. For more information on creating and using GSTs, refer to SAS/C Development System Library Reference.

This option defines the preprocessor symbol _MGST .

This option is ignored when compiling C++ files.

1.174 maplibraries

MapLib

generates a list of hunks by library symbol in the map. The default value is nomaplib. The minimum acceptable abbreviation is mlib.

This option is ignored if you do not specify the link and map options.

1.175 mapxref

MapXreference

writes a symbol cross-reference to the map file that lists each symbol definition and the places each symbol is used. The default value is nomapxreference. The minimum acceptable abbreviation is mxref.

This option can generate a lot of output, but it is useful when you are trying to track down where an unresolved symbol is referenced. This option is ignored if you do not specify the link and map options.

1.176 maxerror

MaximumErrors=n

sets the limit on the number of errors for a single compilation. The default value is 50. The minimum acceptable abbreviation is maxerr.

If a single compilation generates more than n errors, the compiler aborts the compilation. nomaxerr removes any limits.

1.177 maxwarn

```
MaximumWarnings=n
```

sets the limit on the number of warnings for a single compilation. The default value is nomaxwrn. The minimum acceptable abbreviation is maxwrn.

If a single compilation generates more than n warnings, the compiler aborts the compilation. nomaxwrn removes any limits.

1.178 memsize

```
MemorySize=size
```

tells the compiler approximately how much memory you have on your system. You can specify one of the following:

- tiny or t
- small or s
- medium or m
- large or l
- huge or h

The default value is large. The minimum acceptable abbreviation is memsize. This option does not have a negative form.

Larger sizes allow sc to compile more complex programs and to compile faster. Smaller sizes allow sc to continue to work under low-memory conditions. If the compiler runs out of memory during a compilation, it displays the message ***Freeing Resources, attempts to free up memory, and automatically drops to a lower memorysize value.

If you specify link and either memorysize=tiny or memorysize=small, the compiler passes the bufsize=4096 option to the linker.

memorysize affects how and where the compiler stores any debugging information. If the compiler begins to run out of memory, it starts writing debugging information to a disk file. This file is referred to as a debug side file.

memorysize also affects the disposition and buffering of the compiler intermediate file. This option tells the compiler how much initial memory space to reserve for the intermediate information. For large and huge, the intermediate file is kept totally in memory (when the memory is available), which is much faster than writing it to disk. At smaller values, the intermediate file is written to disk, but the memorysize value affects the amount that is buffered.

In addition, if you do not specify the preprocessorbuffer (ppbuf) and/or argumentsize (argsize) options, the memorysize option sets these values for you. If you specify the preprocessorbuffer and/or argumentsize options, the values you specify override the values set

by memorysize.

The following table lists the default values for argumentsize and preprocessorbuffer by memorysize. It also lists the buffer size of the compiler intermediate file.

memsize	Intermediate argsize	ppbuf	Debug Side File Buffer	File Buffer
tiny	127	1024	1024	2K
small	255	2048	4096	8K
medium	511	4096	8192	32K
large	1023	8192	no limit	64K
huge	4800	16384	no limit	128K

See also the descriptions of the preprocessorbuffer and argumentsiz options.

1.179 mcconstants

MultipleCharacterConstants

allows up to four bytes to appear within single quotes as a character constant. This option is included only for compatibility with previous releases of the compiler, and its use is not recommended. The minimum acceptable abbreviation is mccons. The default value is nomccons.

If you specify mccons, a single constant of type int is generated. If fewer than four bytes are provided, they are padded on the left with zeroes, as in the following example:

```
#include <stdio.h>

void main(void)
{
   long l = 'abcd';
   long m = '\x01\x02\x03';
   printf("l=0x%08lx, m=0x%08lx\n", l, m);
}
```

This example program prints the following:

```
l=0x61626364, m=0x00010203
```

This option is ignored when compiling C++ files.

1.180 objectlib

ObjectLibrary=link-library-name

specifies that any resulting object files are to be placed in the named link library. The minimum acceptable abbreviation is objlib.

Do not specify this option if you also specify the link option.

1.181 optalias

OptimizerAlias

disables type-based aliasing assumptions in the optimizer. The default value is nooptimizeralias. The minimum acceptable abbreviation is optalias.

If you specify optimizeralias, the global optimizer uses worst-case aliasing. Specifying optimizeralias can significantly reduce the amount of optimization that can be performed. This option is ignored if you do not specify the optimize and optglobal options.

1.182 optcomp

OptimizerComplexity=n

defines the maximum complexity level of functions to be automatically inlined. The default value is 0. The minimum acceptable abbreviation is optcomp.

The parameter n represents the relative complexity of the function to be inlined and is a count of the number of discrete operations in the function. Try different values for this number until you get the results you want. The higher the number, the more functions you can inline, but the size of your code will grow significantly as well.

If you specify nooptcomp, no complexity-based inlining occurs. This option is ignored if you do not specify the optimize , optglobal , and optimizerinline options.

1.183 optdepth

OptimizerDepth=n

defines the maximum nesting depth of automatically inlined functions. The default value is 0. The minimum acceptable abbreviation is optdep.

This option is ignored if you do not specify the optimize , optglobal , and optimizerinline .

1.184 optimizeglobal

```
OptimizerGlobal

enables the global optimizer.  The default value is optimizerglobal.
The minimum abbreviation is optgo.  This option is ignored if you do
not specify the  optimize  option.
```

1.185 optinline

```
OptimizerInline

allows inlining of functions, including functions defined with the
__inline keyword.  The default value is optimizerinline.  The
minimum acceptable abbreviation is optinl.

This option is ignored if you do not specify the  optimize  and
 optglobal  options.

If you specify nooptinline, the  optinlocal ,  optdepth ,  optcomplexity ,
and  optrdepth  options are ignored.
```

1.186 optinlocal

```
OptimizerInLocal

inlines single-use static functions.  The default value is
nooptimizerinlocal.  The minimum acceptable abbreviation is
optinlocal.

This option is ignored if you do not specify the  optimize ,
 optglobal , and  nooptimizerinline  options.
```

1.187 optloop

```
OptimizerLoop

enables loop optimizations.  The default value is optimizerloop.
The minimum acceptable abbreviation is optloop.

This option is ignored if you do not specify the  optimize  and
 optglobal  options.
```

1.188 optimizepeep

```
OptimizerPeephole

enables the peephole optimizer.  The default value is
optimizerpeephole.  The minimum acceptable abbreviation is optpeep.

This option is ignored if you do not specify the  optimize  option.
```

1.189 optrdepth

```
OptimizerRecurDepth=n

defines the maximum depth of recursion of automatically inlined
functions.  The default value is 0.  The minimum acceptable
abbreviation is optrdep.

This option is ignored if you do not specify the  optimize ,
 optglobal , and  optinline  options.
```

1.190 optsize

```
OptimizerSize

generates smaller code, possibly increasing execution time.  The
default value is nooptsize.  The minimum acceptable abbreviation is
optsize.

This option is ignored if you do not specify the  optimize  and
 optglobal  options.  Do not specify both the optimizersize and
 optimizertime  options.
```

1.191 opttime

```
OptimizerTime

generates code that will run faster but may be larger.  The default
value is nooptimizertime.  The minimum acceptable abbreviation is
opttime.

This option is ignored if you do not specify the optimize and
optglobal options.  Do not specify both the  optimizersize  and
optimizertime options.
```

1.192 optimizeschedule

```
OptimizerSchedule

runs the instruction scheduler.  The default value is nooptschedule.
The minimum acceptable abbreviation is optsched.

The scheduler reorders instructions to make them run faster on
higher-order processors, like the 68040 and 68882, without
sacrificing the ability to run on lower-order processors.  Running
the scheduler may take additional compilation time.

This option is ignored if you do not specify the  optpeep  and
 optimize  options.
```

1.193 parms

```
Parameters=method

indicates how parameters should be passed.  The minimum acceptable
abbreviation is parm.  This option does not have a negative form.

You can specify one of the following:

stack or s
  indicates parameters should be passed on the stack.  The default
  value is stack.

register or r
  indicates parameters should be passed in registers.

both or b
  generates a combination prolog that allows parameters to be passed
  on the stack or in registers.

If your program has prototypes for all routines, you should probably
use parameters=register for increased efficiency.  If you are
placing code into a link library, specify parms=both so that your
library functions accept registerized parameters and parameters that
are passed on the stack.

If you write a function that has the same name as a SAS/C library
function, you need to add the __regargs keyword to your function or
compile with the parms=both or parms=register option.  For example,
you may want to replace the SAS/C library function malloc with your
own version of malloc.  If you compile with the parms=stack option
or define your version of malloc with the __stdargs keyword, then
two versions of malloc are linked into your executable.  If you use
other SAS/C library functions that call malloc, these functions use
the version of malloc in the SAS/C libraries.  However, your
functions that call malloc use your version of malloc.  To make sure
that all calls to malloc are using your version of malloc (including
calls from library routines), define your version with __regargs or
compile with the parms=both or parms=register option.
```

With C++ functions, only functions with "C" linkage obey the value you specify with the parameters option. All C++ functions behave as if you specified parameters=register unless you explicitly declare them with __stdargs or __asm.

For information about passing parameters between C language functions and assembly language functions, refer to Chapter 11, "Using Assembly Language with the C and C++ Languages," in SAS/C Development System User's Guide, Volume 2.

1.194 preprocessonly

PreprocessorOnly

tells the compiler to run only the preprocessor on the source files. The default value is nopreprocessoronly. The minimum acceptable abbreviation is pponly.

If you specify a filename with the objectname option, the compiler writes the output to the file you specify. If you do not specify an output filename, the compiler writes the output to the same filename as the source file but with the extension .p.

This option defines the preprocessor symbol _PPONLY .

1.195 strsect

StringSection=section

indicates where to place strings, data declared static const, and initializers for automatic structures, unions, and arrays.

The minimum acceptable abbreviation is strsect. This option does not have a negative form.

You can specify one of the following:

default
   stores data according to the following rules:

   1. If you specify strmerge, the code section is used.
   2. If you specify data=far or data=near, the named section is used.
   3. Otherwise, the near section is used.

   The default value is default.

near or n
   stores the data in the near data section.

far or f

stores the data in the far data section.

code or c
stores the data in the code section.

If you specify a value other than default, the strings and static const data are placed in the specified section regardless of whether you specify strmerge or data.

This option allows you to merge strings and place them in the far section for residentable programs or shared libraries. Since the strings are read-only, they can be placed safely into far data; placing them in the far section frees up more near data space for your read/write external variables.

The stringsection option has two useful side effects. Because some data are placed in the code section:

- Less data needs to be placed into the near data section. If you have more than 64k of data, you can use the stringsection option to try to reduce the amount of data and allow your code to continue to use the near data model.

- The string constants are addressed relative to the PC (program counter) instead of the beginning of the near data section. Therefore, it is possible to generate programs with no near or far data.

For more information on the code and near data sections, see Chapter 12, "How the Compiler Works"

1.196 structequivalence

StructureEquivalence

tells the compiler not to issue messages if a pointer to one structure type is passed to a function when the function expects a pointer to a different type, if the type passed is equivalent to the type expected. The default value is nostructureequivalence. The minimum acceptable abbreviation is streq.

For information on using equivalent structures, see Chapter 11, "Using SAS/C Extensions to the C and C++ Languages." This option is ignored when compiling C++ files.

1.197 utillib

UtilityLibrary

generates inline calls to the AmigaDOS 2.0 ROM-resident library

utility.library to do integer multiplication and division instead of calling SAS/C library functions to do these operations. The default value is noutilitylibrary. The minimum acceptable abbreviation is utillib.

Specifying utilitylibrary makes your executable smaller and faster by taking advantage of 68020 instructions if they are available, but your program runs only under AmigaDOS 2.0. If you link using the sc command and the utilitylibrary and link options, sc defines all the SAS/C library integer conversion stub routines to stubs that call utility.library. This action prevents the SAS/C library routines that use integer conversion routines from using the SAS/C versions of the routines. If you link using the slink command, you need to specify the following define linker options:

```
define __CXM33=__UCXM33
define __CXD33=__UCXD33
define __CXM22=__UCXM22
define __CXD22=__UCXD22
```

These define options are in the file LIB:utillib.with, so you can link with the following command:

```
slink with LIB:utillib.with options
```

1.198 xrefheaders

XReferenceHeaders

generates a cross-reference of user header files. The default value is xreferenceheaders. The minimum acceptable abbreviation is xhead.

This option is ignored if you do not specify the xreference option.

1.199 xrefsystem

XReferenceSystem

includes symbols defined in system header files in the cross-reference. The default value is xreferencesystem. The minimum acceptable abbreviation is xsys.

This option is ignored if you do not specify the xreference option.